Graphs

Graphs ↔ Networks

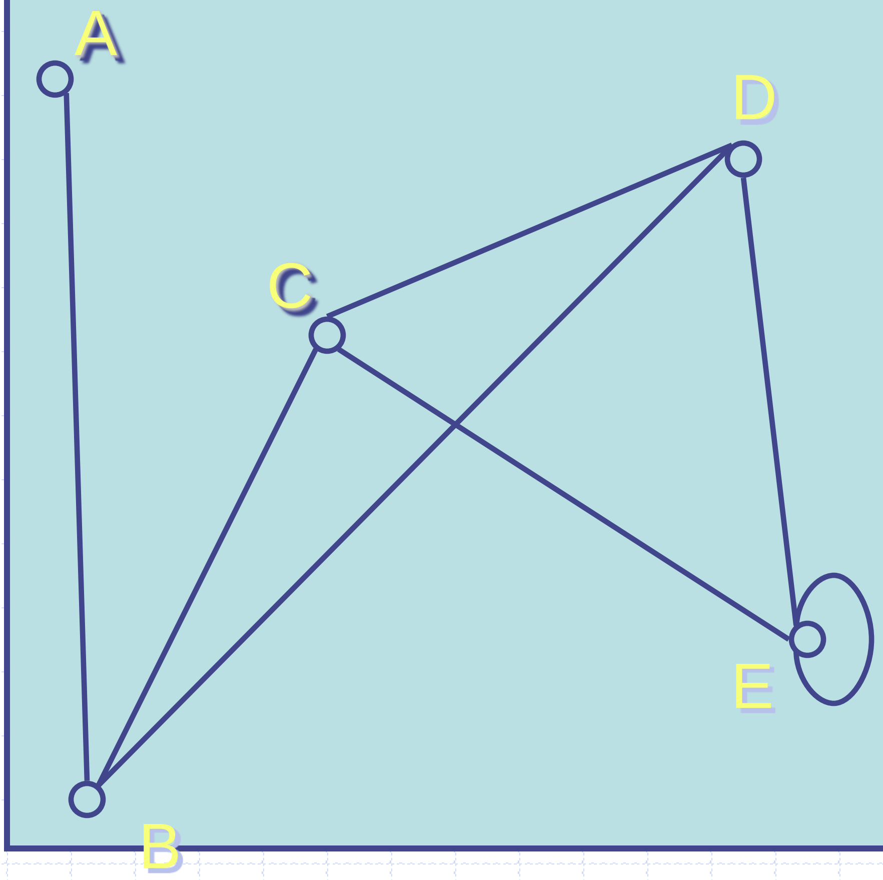
Graph (Network)	Vertices (Nodes)	Edges (Arcs)	Flow
Communications	Telephones exchanges, computers, satellites	Cables, fiber optics, microwave relays	Voice, video, packets
Circuits	Gates, registers, processors	Wires	Current
Mechanical	Joints	Rods, beams, springs	Heat, energy
Hydraulic	Reservoirs, pumping stations, lakes	Pipelines	Fluid, oil
Financial	Stocks, currency	Transactions	Money
Transportation	Airports, rail yards, street intersections	Highways, railbeds, airway routes	Freight, vehicles, passengers

Graphs consist of

- points called *vertices*
- lines called *edges*

1. Edges connect *two* vertices.
2. Edges only intersect at vertices.
3. Edges joining a vertex to itself are called *loops.*

Example : List its vertices and edges.



- One graph may be drawn in (infinitely) many ways, but it always provides us with the same information
- Graphs are ***a structure for describing relationships*** between objects
  (The vertices denote the objects and the edges represent the relationship)
- Mathematically, graphs are binary-relations on their vertex set (except for multigraphs)
- The magnitude of graph *G* is characterized by number of vertices |V| (called the order of *G*) and number of edges |E| (size of *G*)
- The running time of algorithms are measured in terms of the order and size

Simple Graphs

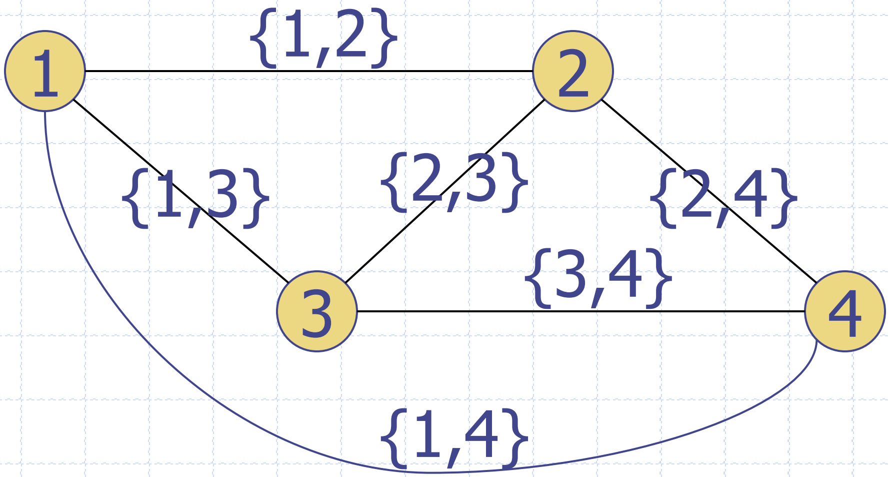
Different purposes require different types of graphs.

e.g. Suppose a local computer network

- Is bidirectional (undirected)
- Has no loops (no "self-communication")
- Has unique connections between computers

Sensible to represent as follows:

Simple Graphs



- Vertices are labeled to associate with particular computers
- Each edge can be viewed as the set of its two endpoints

Simple Graphs

DEF: A ***simple graph*** $G = (V,E)$ consists of a non-empty set V of ***vertices*** (or ***nodes***) and a set E (possibly empty) of ***edges*** where each edge is a subset of V with cardinality 2 (an unordered pair)

Simple Graphs

Q: For a set V with n elements, how many edges are possible?

A: The number of pairs in V

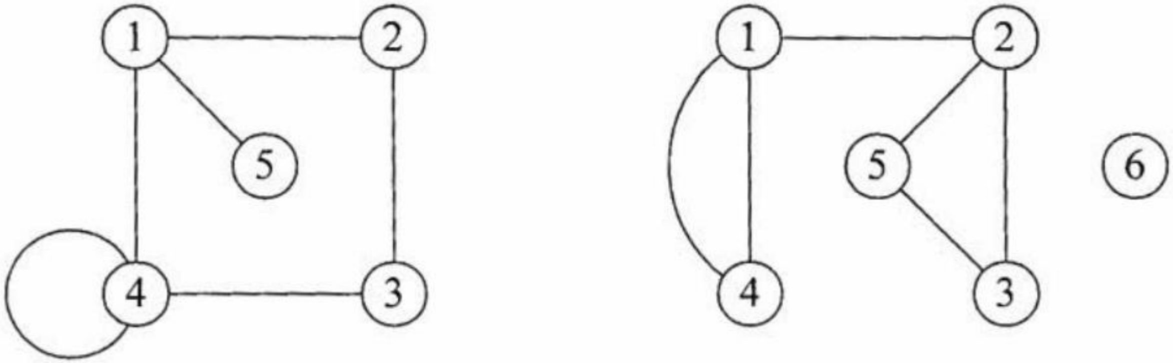
$$= C(n,2) = n \cdot (n-1) / 2$$

Simple Graphs

Q: How many possible graphs are there for the same set of vertices V?

A: The number of subsets in the set of possible edges. There are $n \cdot (n-1) / 2$ possible edges, therefore the number of graphs on V is $2^{n(n-1)/2}$

Q: If following Graphs are simple?



(a) (b)

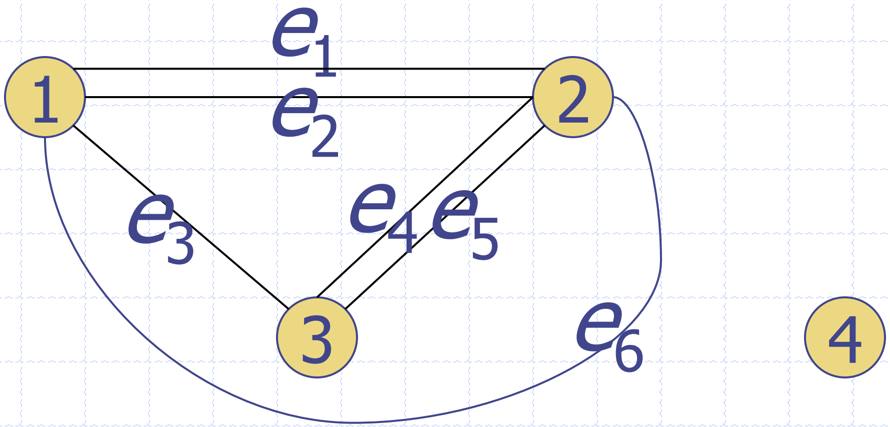
A: No

A graph **with no loops** and **no parallel edges** is called a simple graph.

Multigraphs

If computers are connected via internet instead of directly, there may be several routes to choose from for each connection. Depending on traffic, one route could be better than another. Makes sense to allow multiple edges, but still no self-loops:

Multigraphs



Edge-labels distinguish between edges sharing same endpoints. Labeling can be thought of as function:

e_1 → {1,2}, e_2 → {1,2}, e_3 → {1,3}, e_4 → {2,3}, e_5 → {2,3}, e_6 → {1,2}

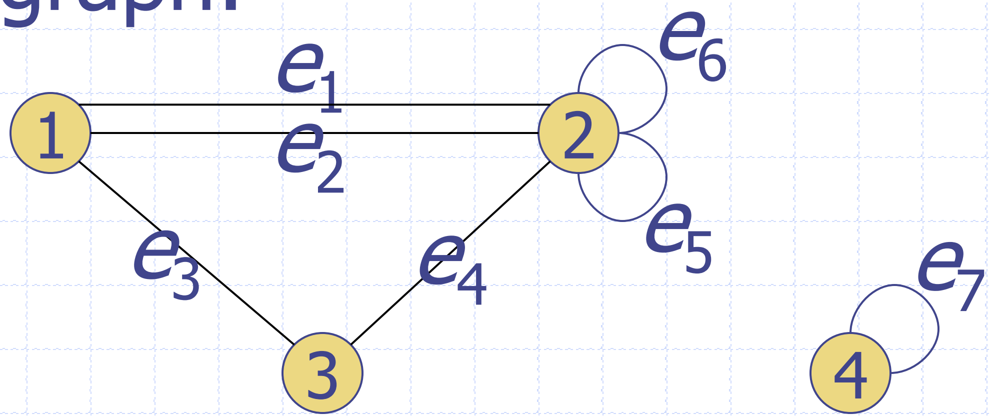
Multigraphs

DEF: A ***multigraph*** $G = (V,E,f)$ consists of a non-empty set V of ***vertices*** (or ***nodes***), a set E (possibly empty) of ***edges*** and a function f with domain E and codomain the set of pairs in V.

- No loops
- More than one edge can join two vertices – multiple edges or parallel edges

Pseudographs

- If self-loops are allowed we get a pseudograph:



Now edges may be associated with a single vertex, when the edge is a loop

e_1 → {1,2}, e_2 → {1,2}, e_3 → {1,3},

e_4 → {2,3}, e_5 → {2}, e_6 → {2}, e_7 → {4}

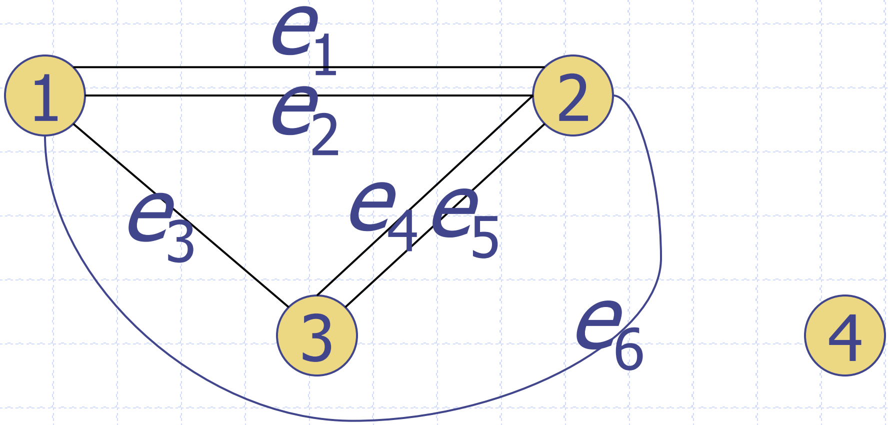
Pseudographs

DEF: A ***pseudograph*** $G = (V,E,f)$ consists of a non-empty set V of ***vertices*** (or ***nodes***), a set E (possibly empty) of ***edges*** and a function f with domain E and codomain the set of pairs and singletons in V.

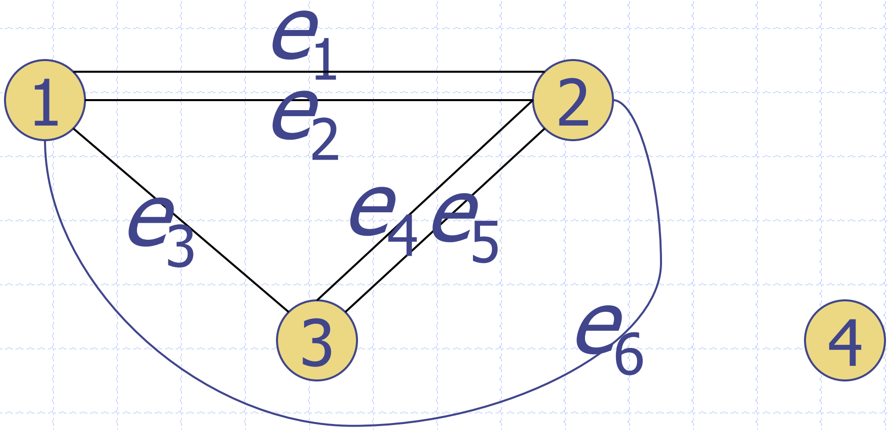
- i.e. A multigraph with loops

Undirected Graphs Terminology

- Vertices are ***adjacent*** if they are the endpoints of the same edge.



Undirected Graphs Terminology



Q: Which vertices are adjacent to 1? How about adjacent to 2, 3, and 4?

A:1 is adjacent to 2 and 3

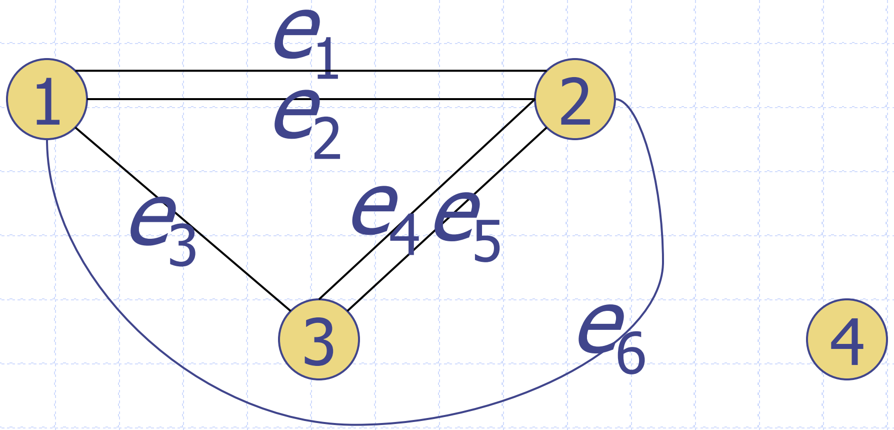
2 is adjacent to 1 and 3

3 is adjacent to 1 and 2

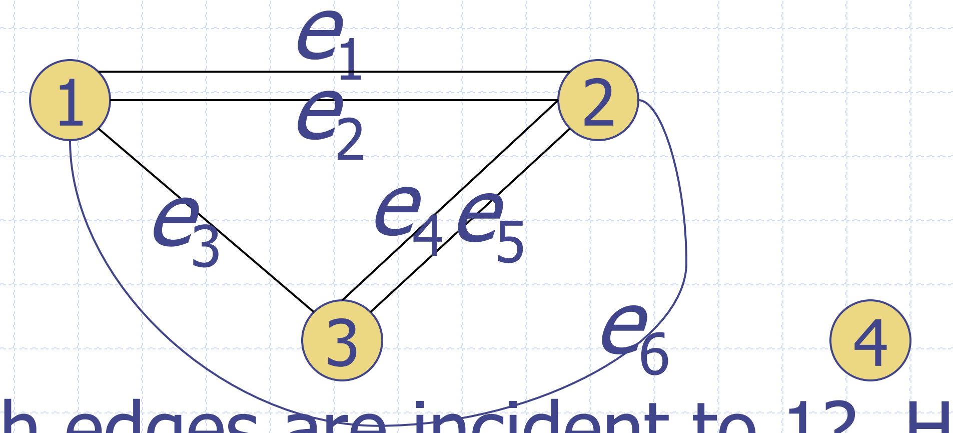
4 is not adjacent to any vertex➔ Isolated Vertex

Undirected Graphs Terminology

A vertex is ***incident*** with an edge (and the edge is incident with the vertex) if it is the endpoint of the edge.



Undirected Graphs Terminology



Q: Which edges are incident to 1? How about incident to 2, 3, and 4?

A: e_1, e_2, e_3, e_6 are incident with 1

2 is incident with e_1, e_2, e_4, e_5, e_6

3 is incident with e_3, e_4, e_5

4 is not incident with any edge

Finite & Infinite Graphs

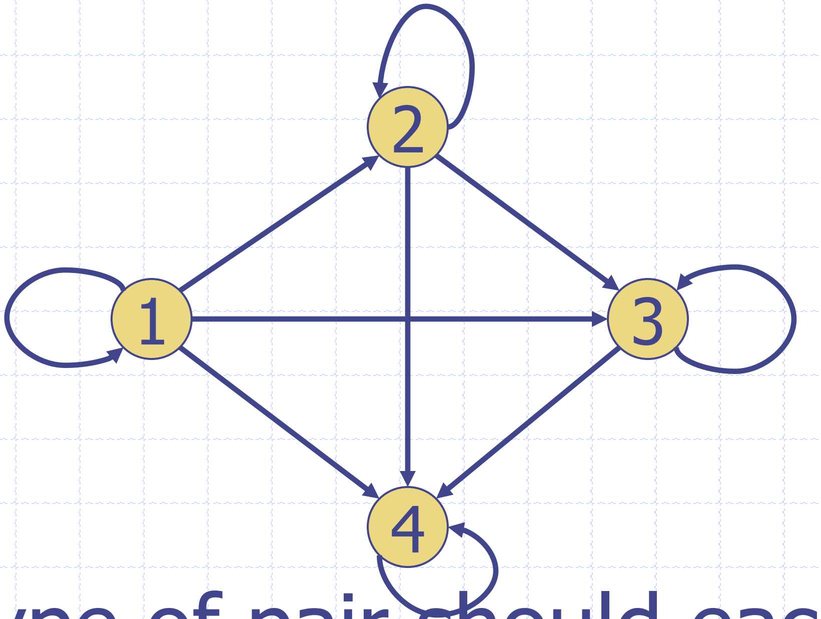
- A Graph with finite number of vertices as well as finite number of edges is called as Finite Graph

Else

is called as Infinite Graph

Digraphs

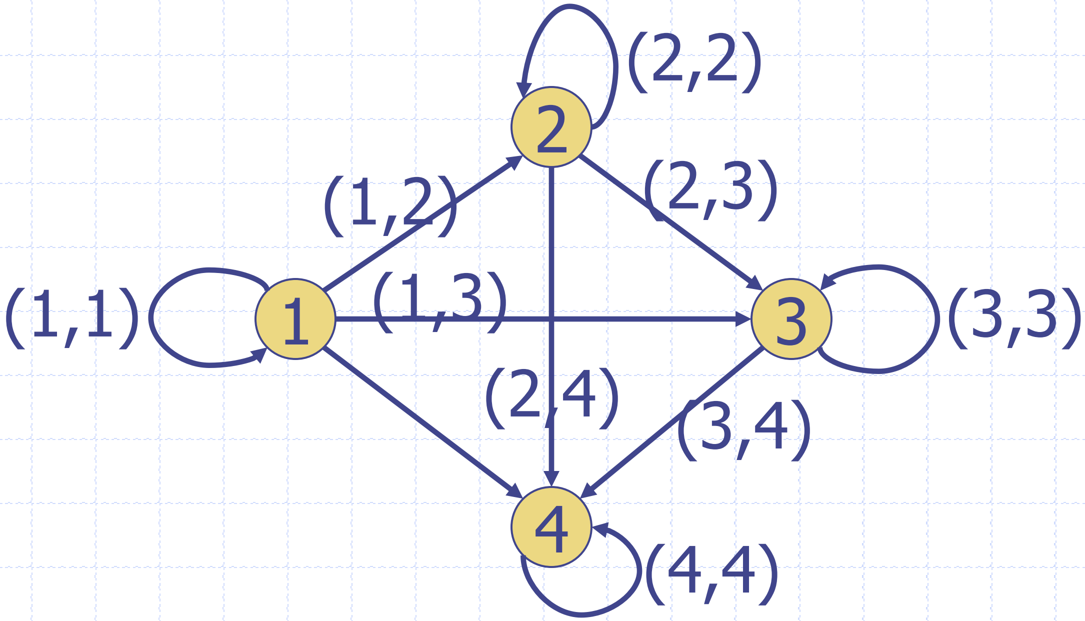
Digraphs for representing relations:



Q: What type of pair should each edge be (multiple edges not allowed)?

Digraphs

A: Each edge is directed so an *ordered* pair (or tuple) rather than unordered pair.



Thus the set of edges *E* is just the represented relation on *V*.

Digraphs

DEF: A ***directed graph*** (or ***digraph***) $G = (V,E)$ consists of a non-empty set V of ***vertices*** (or ***nodes***) and a set E of ***edges*** with $E \subseteq V \times V$.

The edge (a,b) is also denoted by $a \rightarrow b$ and a is called the ***source*** of the edge while b is called the ***target*** of the edge.

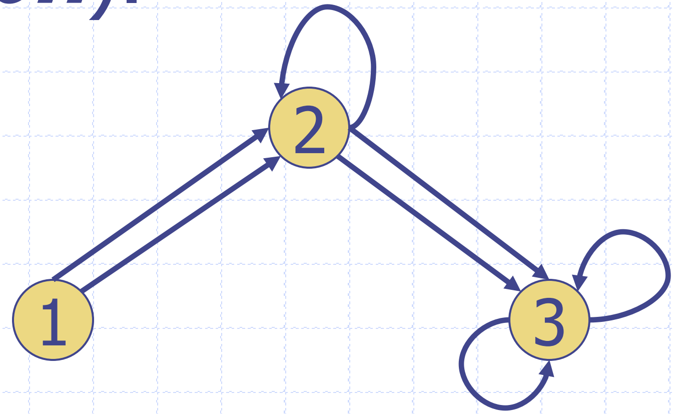
Digraphs

Q: For a set V with n elements, how many possible digraphs are there?

A: The same as the number of relations on V, which is the number of subsets of $V \times V$ so possible diagraphs are: $2^{n \cdot n}$

Directed Multigraphs

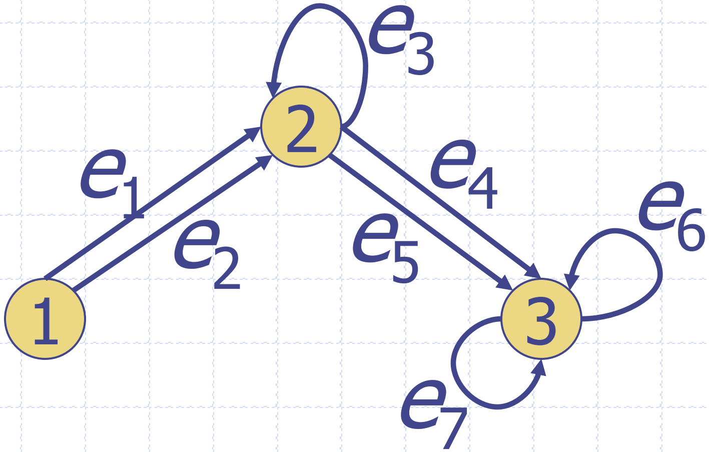
If also want to allow multiple edges in a digraph, get a ***directed multigraph*** (or ***multi-digraph***).



Q: How to use sets and functions to deal with multiple directed edges, loops?

Directed Multigraphs

A: Have function with domain the edge set and codomain $V \times V$.



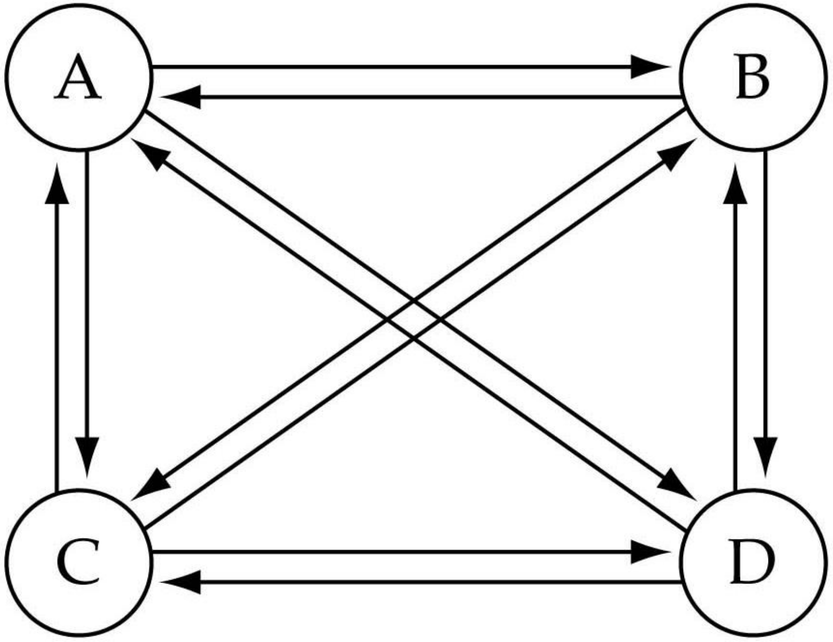
$e_1 \rightarrow (1,2)$, $e_2 \rightarrow (1,2)$, $e_3 \rightarrow (2,2)$, $e_4 \rightarrow (2,3)$, $e_5 \rightarrow (2,3)$, $e_6 \rightarrow (3,3)$, $e_7 \rightarrow (3,3)$

Directed Graph

- What is the number of edges in a complete directed graph with N vertices?

*N * (N-1)*

$O(N^2)$



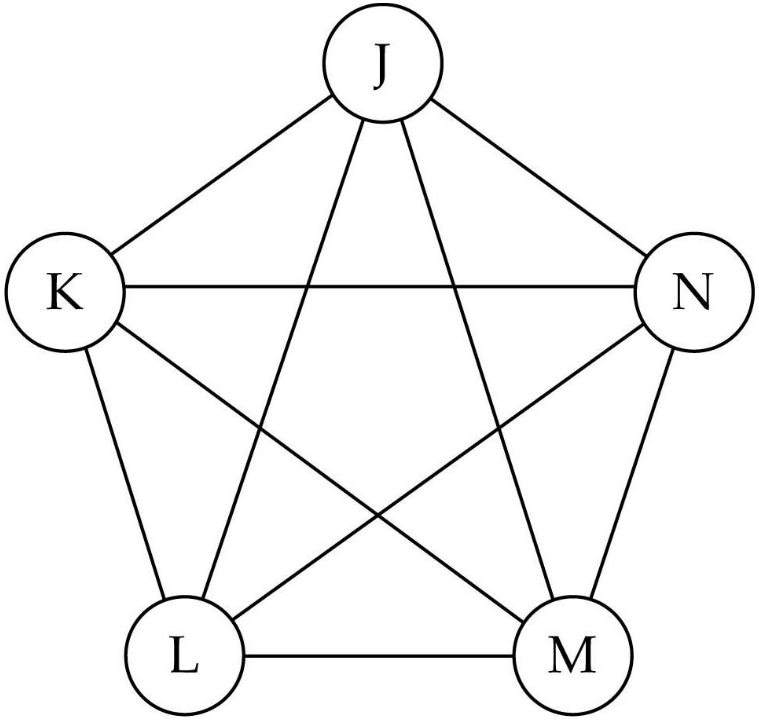
(a) Complete directed graph.

Undirected Graph

- What is the number of edges in a complete undirected graph with N vertices?

*N * (N-1) / 2*

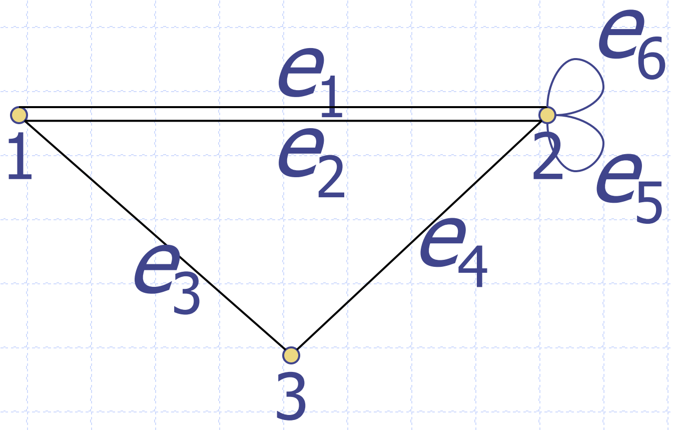
$O(N^2)$



(b) Complete undirected graph.

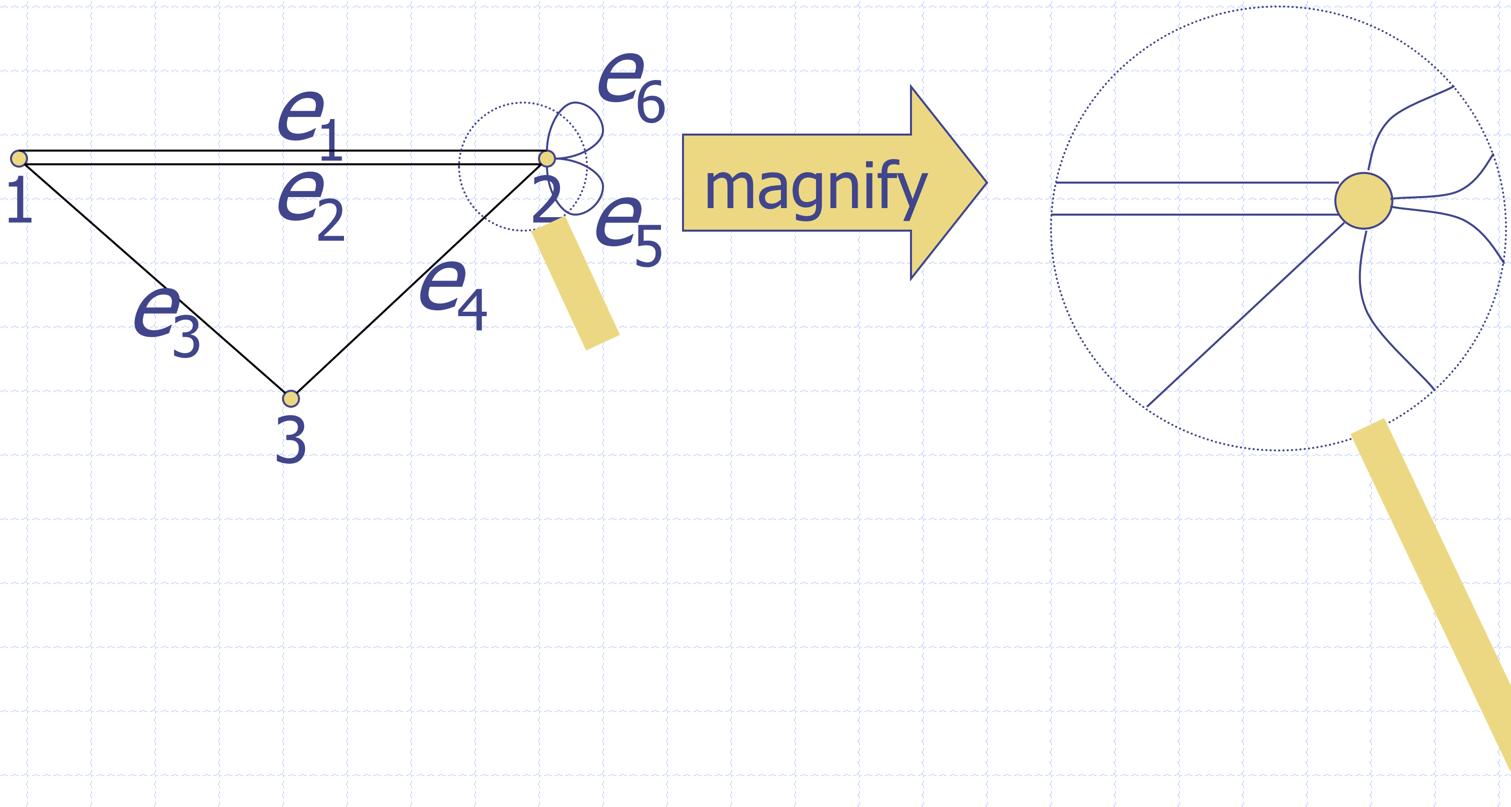
Degree

- The number of edges incident on a vertex; self loops counted twice, is called as the ***degree*** of a vertex



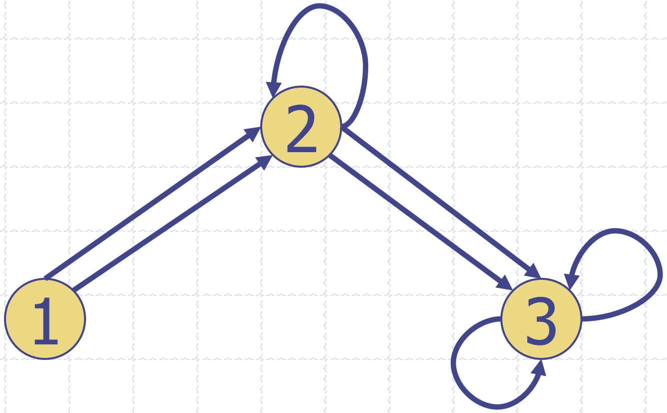
Degree

Thus deg(2) = 7 even though 2 only incident with 5 edges.



Oriented Degree when Edges Directed

The ***in-degree*** of a vertex (deg$^-$) counts the number of edges that stick *in* to the vertex. The ***out-degree*** (deg$^+$) counts the number sticking *out*.



Q: What are in-degrees and out-degrees of all the vertices?

Oriented Degree when Edges Directed

A: $\deg^-(1) = 0$

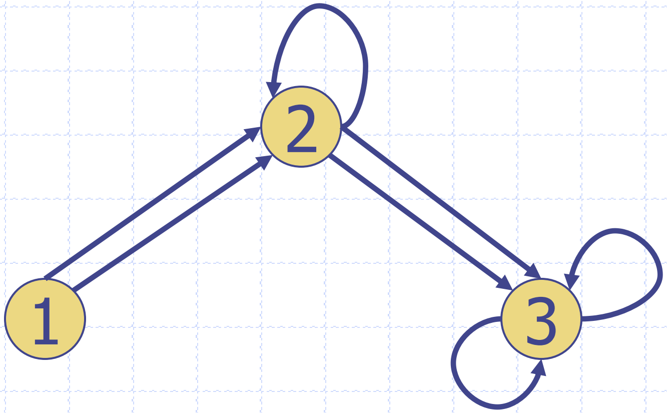
$\deg^-(2) = 3$

$\deg^-(3) = 4$

$\deg^+(1) = 2$

$\deg^+(2) = 3$

$\deg^+(3) = 2$



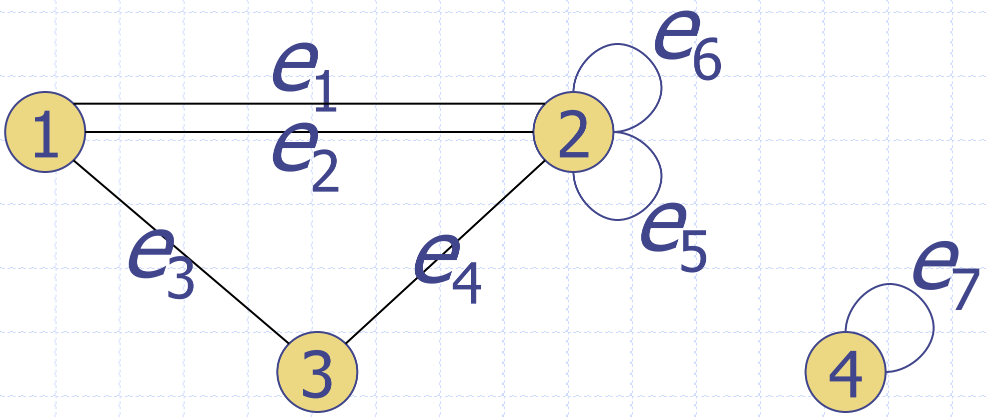
Vertex 2 loop degree =2

= 1 indegree+1 outdegree

- A vertex having no incident edge is called as isolated vertex.
- A vertex of degree one, is called as pendent vertex or an end vertex.
- Total degree of the vertex =

  in degree + out degree
- Source = A vertex with zero in degree
- Sink = A vertex with zero out degree

Handshaking Theorem



There are two ways to count the number of edges in the above graph:

1. Just count the set of edges: 7
2. Count *seeming* edges vertex by vertex and divide by 2 because double-counted edges: (deg(1)+deg(2)+deg(3)+deg(4))/2 = (3+7+2+2)/2 = 14/2 = 7

Handshaking Theorem

- THM: In an undirected graph

$$|E| = \frac{1}{2}\sum_{e\in E}\deg(e)$$

In a directed graph

$$|E| = \sum_{e\in E}\deg^{+}(e) = \sum_{e\in E}\deg^{-}(e)$$

Handshaking Theorem

Lemma: The number of vertices of odd degree must be even in an undirected graph.

Proof : Otherwise would have

$2|E|$ = Sum of even no.'s

+ an odd number of odd no.'s

➔ even = even + odd

–this is impossible. □

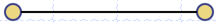
Summary: Types of Graph & its Structure

Type	Edges	Multiple Edges Allowed?	Loops Allowed?
Simple Graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Psuedograph	Undirected	Yes	Yes
Simple Directed Graph	Directed	No	No
Directed Multigraph	Directed	Yes	Yes
Mixed Graph	Directed and Undirected	Yes	Yes

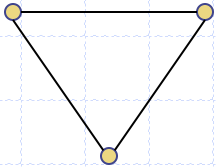
Graph Patterns
Complete Graphs - K_n

A simple graph is ***complete*** if every pair of distinct vertices share an edge. The notation K_n denotes the complete graph on n vertices.

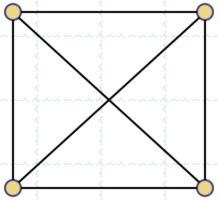
K_1



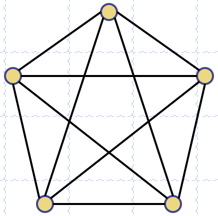
K_2



K_3



K_4

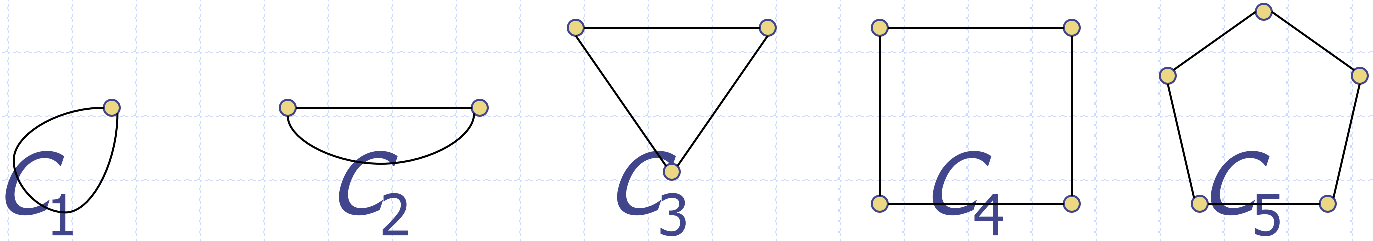


K_5

Graph Patterns

Cycles - C_n

- The ***cycle graph*** C_n is a circular graph with $V = \{0,1,2,\ldots,n-1\}$ where vertex i is connected to $i+1$ **mod** n and to $i-1$ **mod** n. They look like polygons:

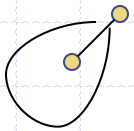


Q: What type of graph are C_1 and C_2 ?

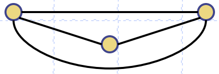
A: C1=Loop and C2=Multigraph ➔Pseudographs

Graph Patterns
Wheels - W_n

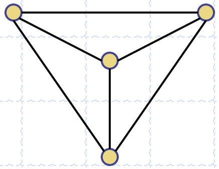
- The ***wheel graph*** W_n is just a cycle graph with an extra vertex in the middle:



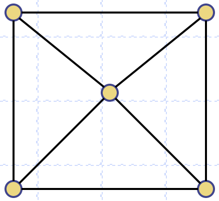
W_1



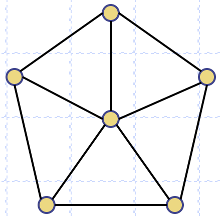
W_2



W_3



W_4



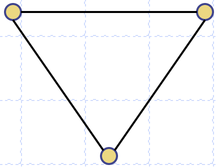
W_5

Usually consider wheels with 3 or more spokes only.

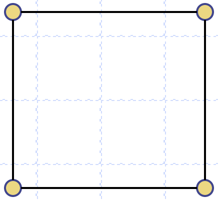
Graph Patterns
Regular Graphs - R_n

The ***Regular graph*** R_n in which all vertices are of equal degree.

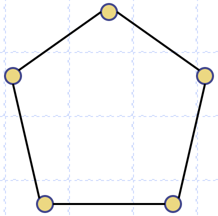
Following are Regular Graphs with degree 2



R_3



R_4

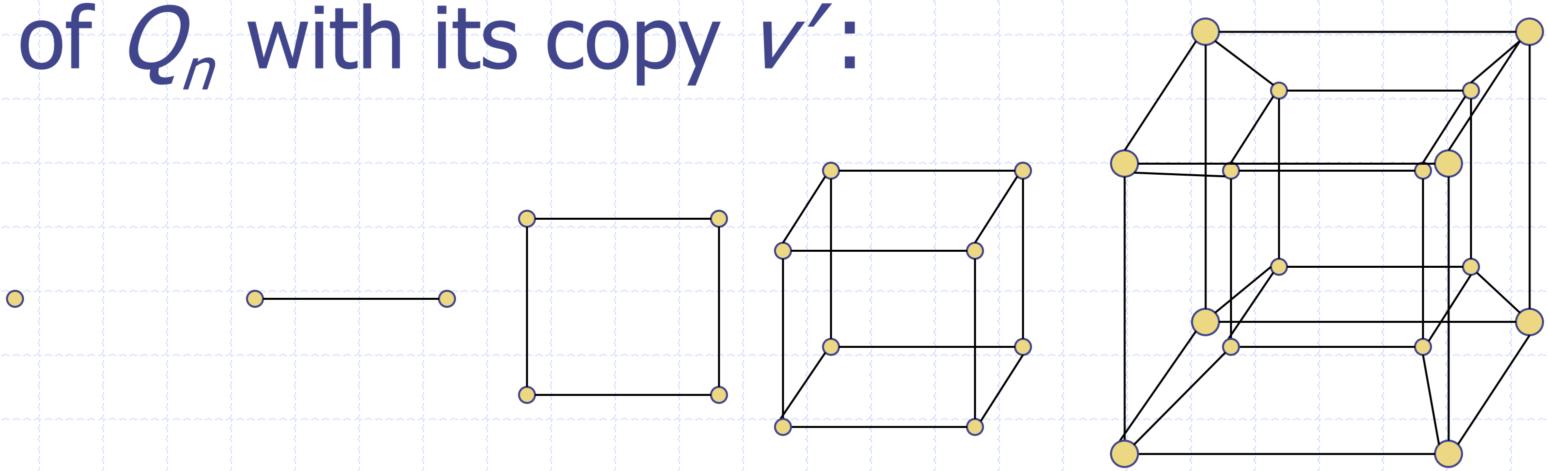


R_5

- Every Null Graph is Regular with degree 0
- Every Complete Graph K_n is Regular with degree n-1
- If G has n vertices with regular degree r→ G has ½(r.n) edges

Graph Patterns
Cubes - Q_n

- The *n-**cube*** Q_n is defined recursively. Q_0 is just a vertex. Q_{n+1} is obtained by taking 2 copies of Q_n and joining each vertex v of Q_n with its copy v':

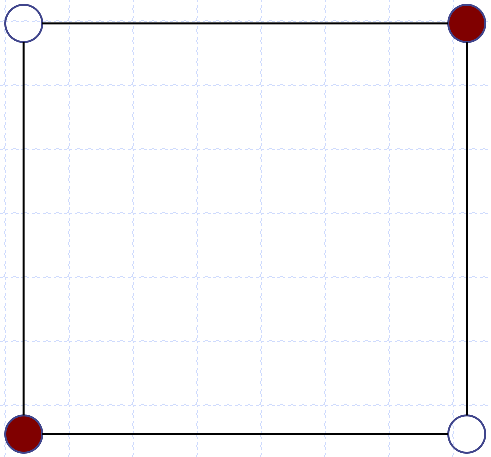


Bipartite Graphs

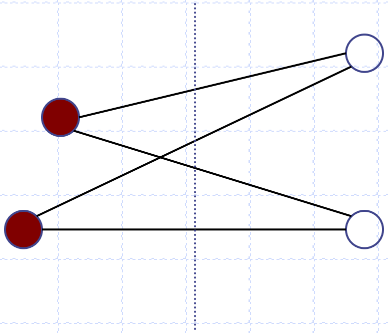
A simple graph is ***bipartite*** if V can be partitioned into $V = V_1 \cup V_2$ so that any two adjacent vertices are in different parts of the partition. Another way of expressing the same idea is ***bichromatic***: vertices can be colored using two colors so that no two vertices of the same color are adjacent.

Bipartite Graphs

EG: C_4 is a bichromatic:

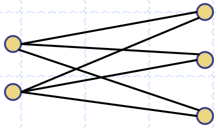


And so is bipartite, if we redraw it:

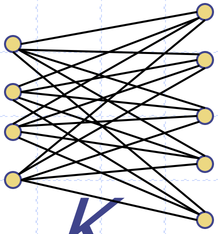


Graph Patterns Complete Bipartite - $K_{m,n}$

When all possible edges exist in a simple bipartite graph with *m* red vertices and *n* green vertices, the graph is called ***complete bipartite*** and the notation $K_{m,n}$ is used. e.g:

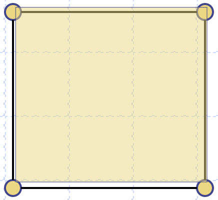
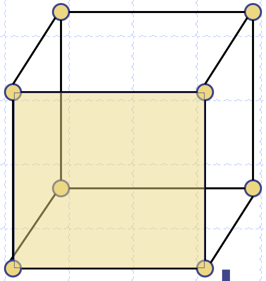


$K_{2,3}$



$K_{4,5}$

Subgraphs

Notice that the 2-cube  occurs inside the 3-cube 

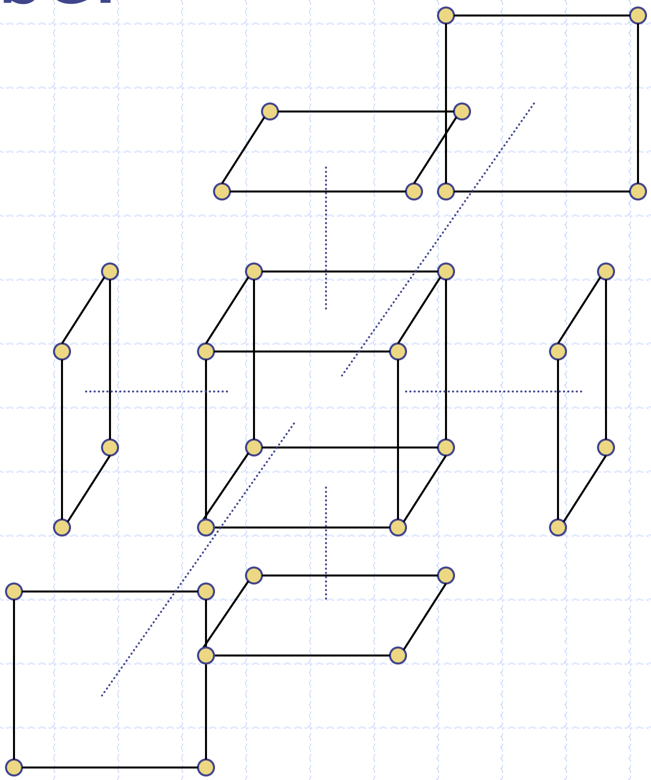
In other words, Q_2 is a subgraph of Q_3 :

DEF: Let $G = (V,E)$ and $H = (W,F)$ be graphs. H is said to be a ***subgraph*** of G, if $W \subseteq V$ and $F \subseteq E$.

Q: How many Q_2 subgraphs does Q_3 have?

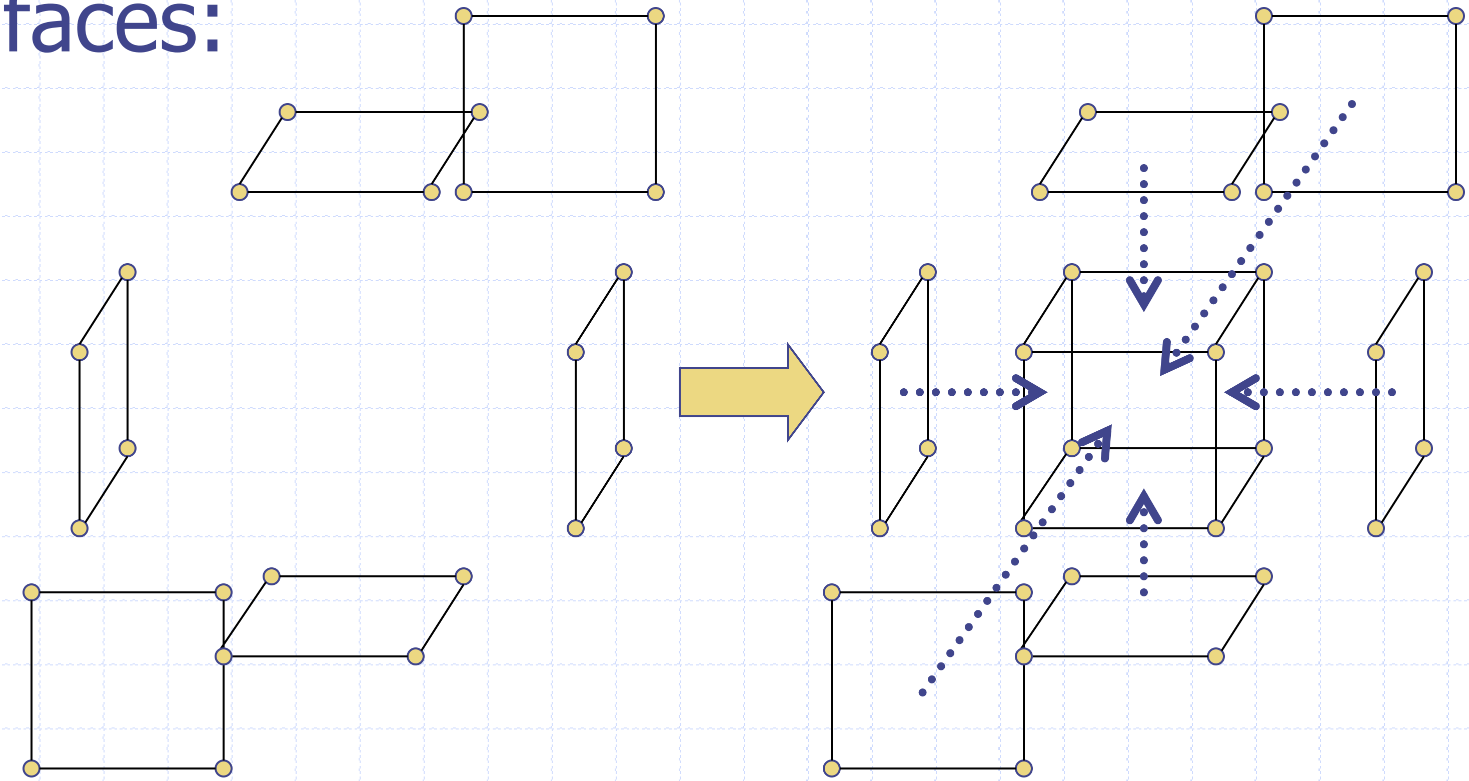
Subgraphs

- A: Each face of Q_3 is a Q_2 subgraph so the answer is 6, as this is the number of faces on a 3-cube:



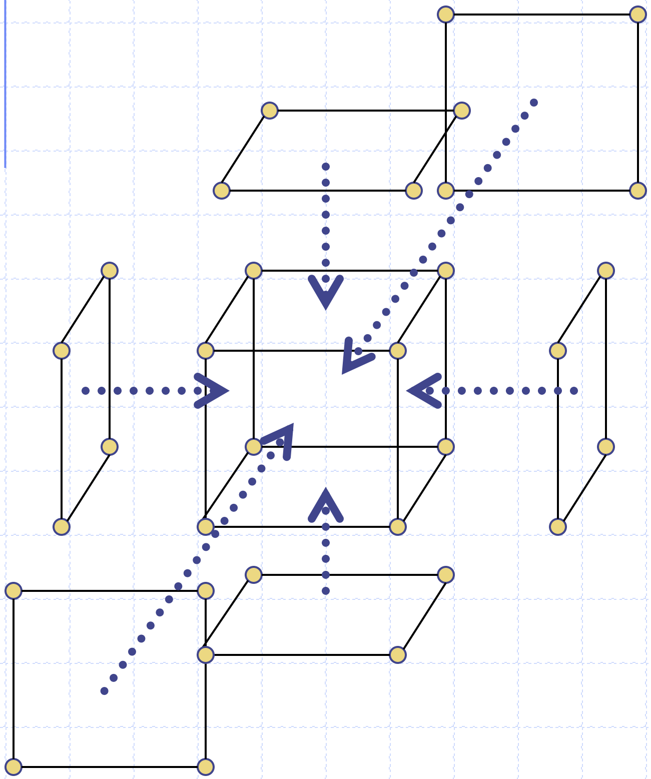
Unions

- In previous example can actually reconstruct the 3-cube from its 6 (2-cube) faces:

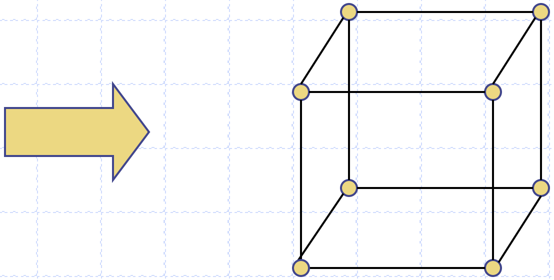


Unions

- If we assign the 2-cube faces the names S_1, S_2, S_3, S_4, S_5, S_6 then Q_3 is the union of its faces:



$$Q_3 = S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5 \cup S_6$$

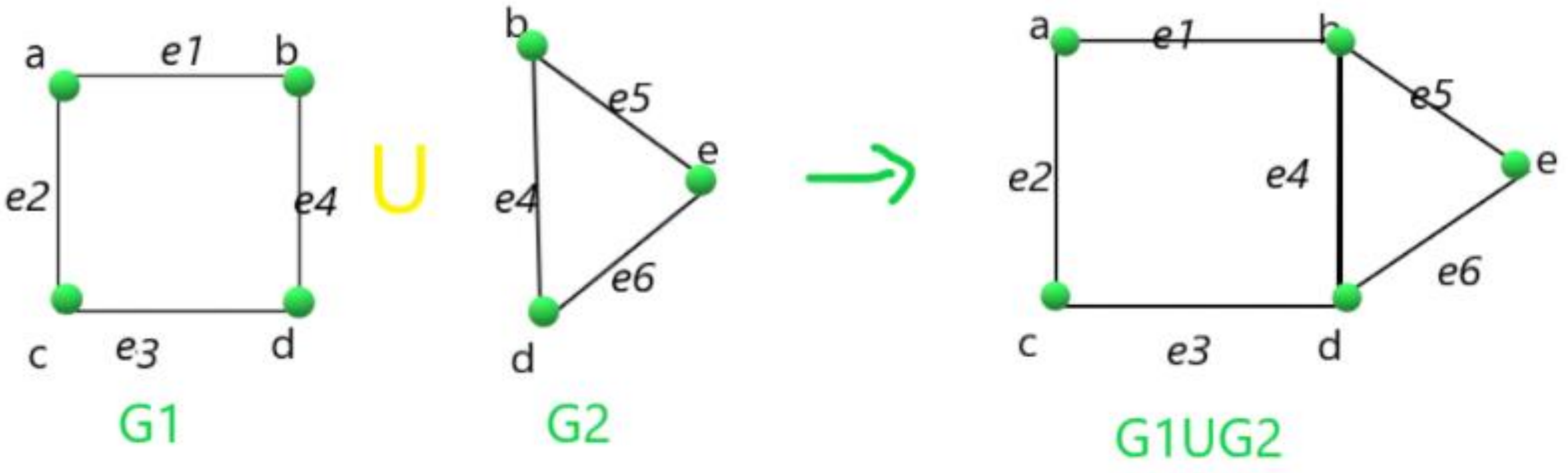


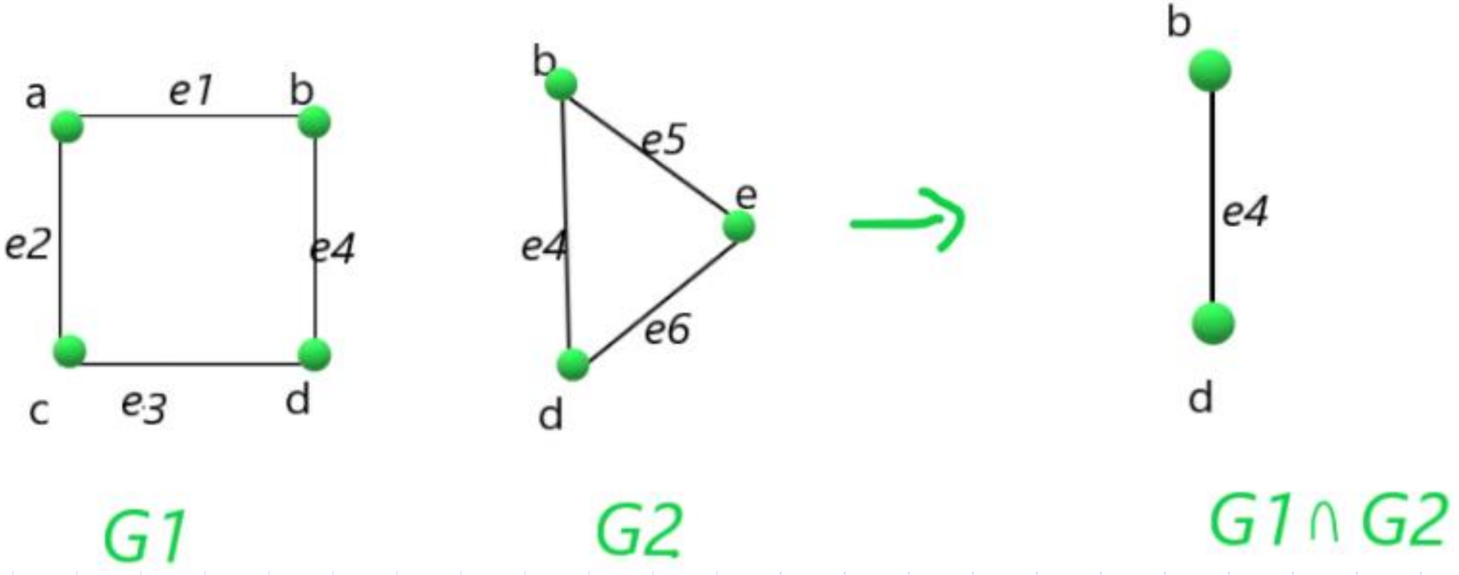
Unions

DEF: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two simple graphs (and V_1, V_2 may or may not be disjoint). The ***union*** of G_1, G_2 is formed by taking the union of the vertices and edges. i.e. $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$.

A similar definitions can be created for unions of digraphs, multigraphs, pseudographs, etc.

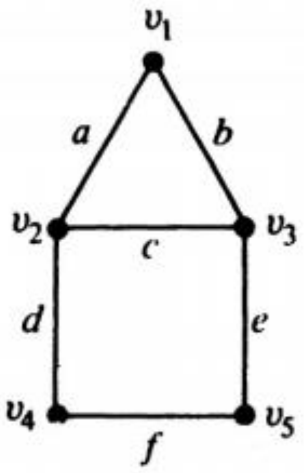
Union and Intersection



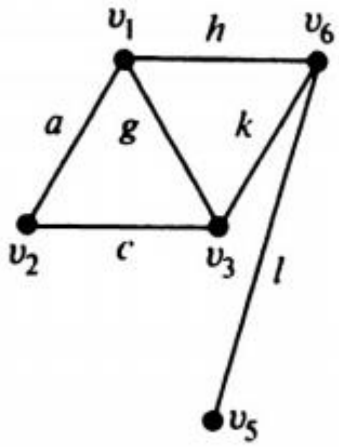


Ring Sum of Graphs

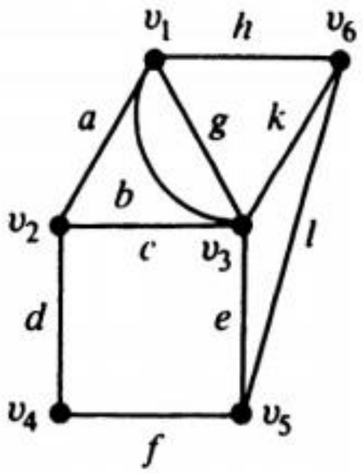
- The ring sum of two graphs G1 and G2 (written as $G1 \oplus G2$) is a graph consisting of the vertex set $V1 \cup V2$ and of edges that are either in G1 or G2 , but not in both



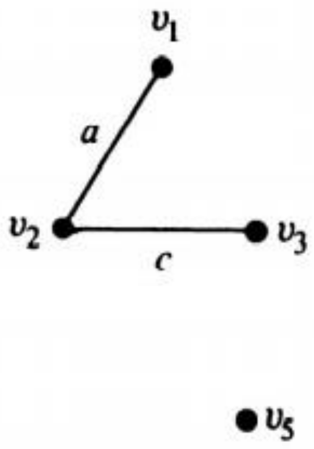
G_1



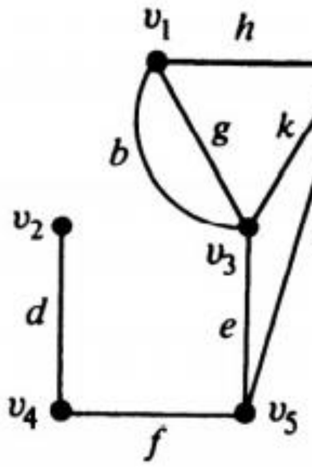
G_2



$G_1 \cup G_2$



$G_1 \cap G_2$



$G_1 \oplus G_2$

Properties of a Graph

- Commutative operations:

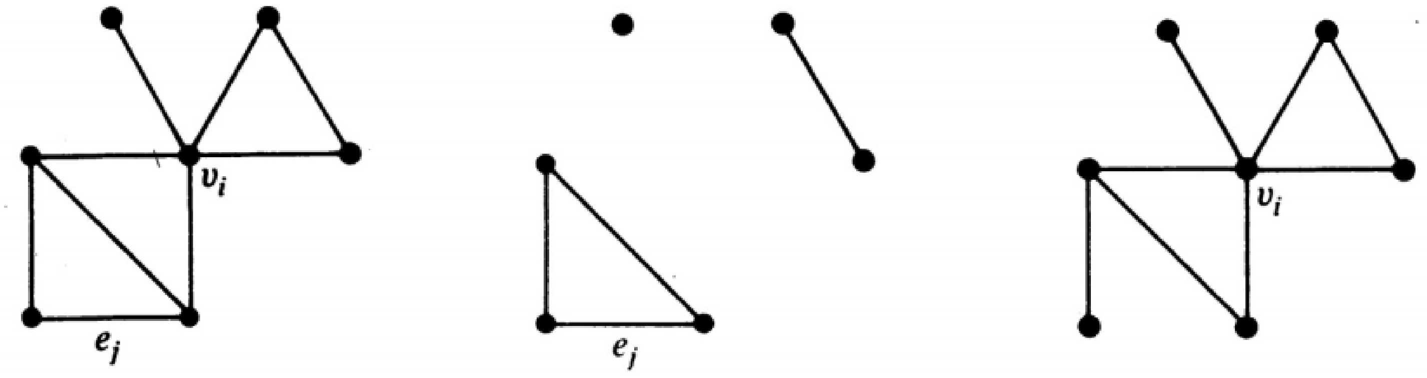
$$G_1 \bigcup G_2 = G_2 \bigcup G_1, \qquad G_1 \bigcap G_2 = G_2 \bigcap G_1,$$
$$G_1 \oplus G_2 = G_2 \oplus G_1.$$

- If G1 and G2 are edge disjoint, then G1 ∩ G2 is a null graph , and G1 ⊕ G2 = G1 ∪ G2
- If G1 and G2 are vertex disjoint, then G1 ∩ G2 is empty
- For any graph G; G ∪ G = G ∩ G = G,

  and G ⊕ G = a null graph

Deletion of a vertex or an edge

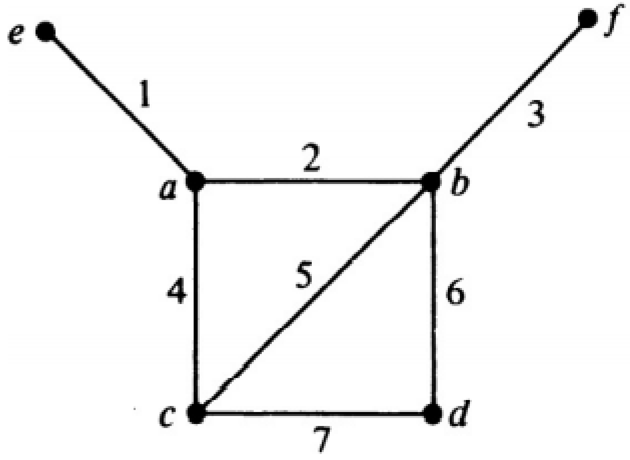
- If vi is a vertex in graph G, then G — vi denotes a subgraph of G obtained by deleting (i.e., removing) vi from G → Deletion of a vertex always implies the deletion of all edges incident on that vertex
- If ej is an edge in G, then G — ej is a subgraph of G obtained by deleting ej from G → Deletion of an edge does not imply deletion of its end vertices

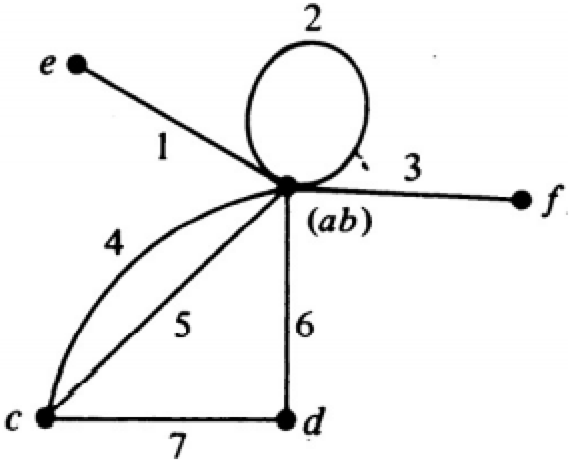


G　　　　$(G - v_i)$　　　　$(G - e_j)$

Fusion of vertices

- A pair of vertices a, b in a graph are said to be fused (merged or identified) if the two vertices are replaced by a single new vertex such that every edge that was incident on either a or b or on both is incident on the new vertex. →Thus fusion of two vertices does not alter the number of edges, but it reduces the number of vertices by one

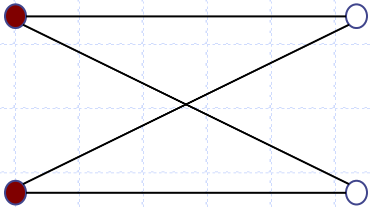




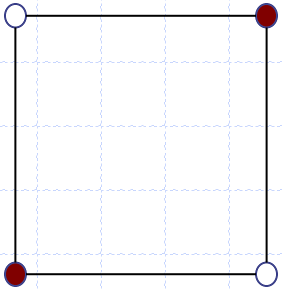
Graph Isomorphism

- Intuitively, two graphs are isomorphic if can bend, stretch and reposition vertices of the first graph, until the second graph is formed.

e.g. Can twist or re-label:



to obtain:



Graph Isomorphism Undirected Graphs

DEF: Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are pseudographs. Let $f : V_1 \rightarrow V_2$ be a function s.t.:

1) f is bijective *(one-to-one correspondence, or invertible)*
2) for all vertices u,v in V_1, the number of edges *between* u and v in G_1 is the same as the number of edges between $f(u)$ and $f(v)$ in G_2.

Then f is called an ***isomorphism*** and G_1 is said to be ***isomorphic*** to G_2.

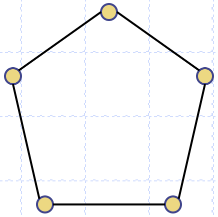
Graph Isomorphism Digraphs

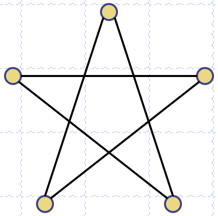
DEF: Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are directed multigraphs. Let $f : V_1 \rightarrow V_2$ be a function s.t.:

1) f is bijective
2) for all vertices u,v in V_1, the number of edges *from* u to v in G_1 is the same as the number of edges between $f(u)$ and $f(v)$ in G_2.

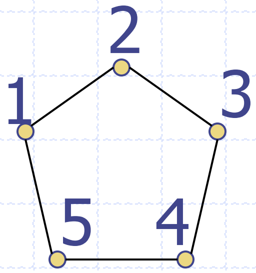
Then f is called an ***isomorphism*** and G_1 is said to be ***isomorphic*** to G_2.

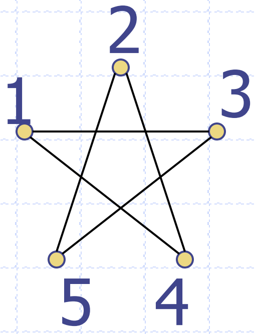
Graph Isomorphism -Example

Prove that 

is isomorphic to 

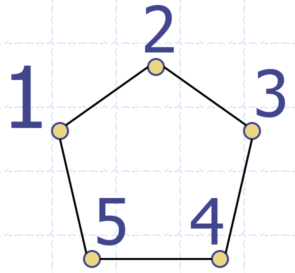
First label the vertices:

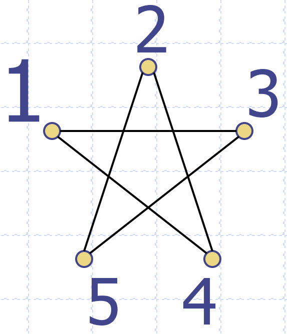




Graph Isomorphism -Example

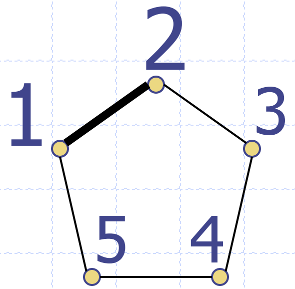
- Next, set $f(1) = 1$ and try to walk around clockwise on the star.

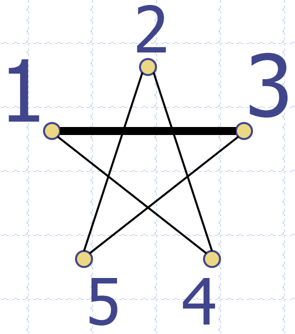




Graph Isomorphism -Example

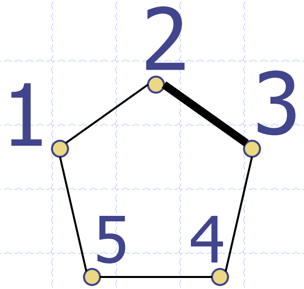
- Next, set $f(1) = 1$ and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set $f(2) = 3$.

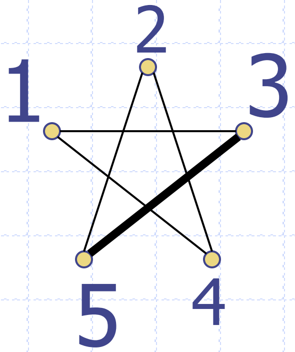




Graph Isomorphism -Example

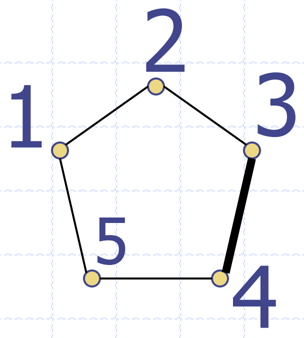
- Next, set $f(1) = 1$ and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set $f(2) = 3$. Next vertex is 5 so set $f(3) = 5$.

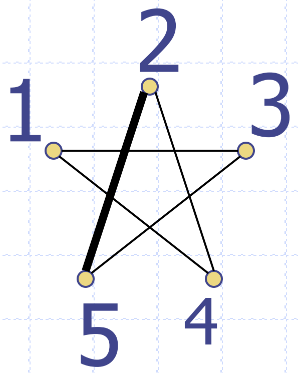




Graph Isomorphism -Example

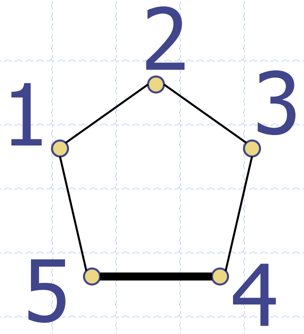
- Next, set *f* (1) = 1 and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set *f* (2) = 3. Next vertex is 5 so set *f* (3) = 5. In this fashion we get *f* (4) = 2

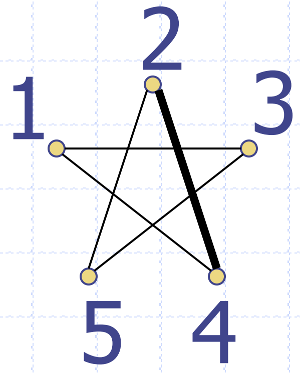




Graph Isomorphism -Example

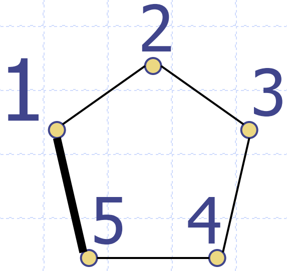
- Next, set $f(1) = 1$ and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set $f(2) = 3$. Next vertex is 5 so set $f(3) = 5$. In this fashion we get $f(4) = 2$, $f(5) = 4$.

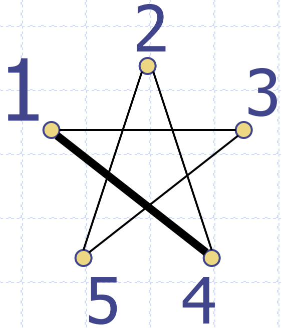




Graph Isomorphism -Example

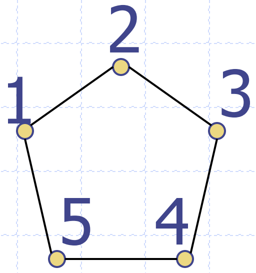
- Next, set *f*(1) = 1 and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set *f*(2) = 3. Next vertex is 5 so set *f*(3) = 5. In this fashion we get *f*(4) = 2, *f*(5) = 4. If we would continue, we would get back to *f*(1) =1 so this process is well defined and *f* is a morphism.

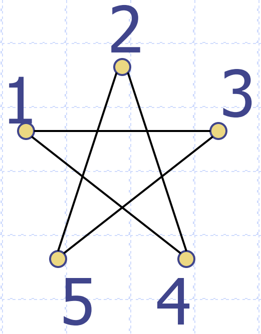




Graph Isomorphism -Example

Next, set *f*(1) = 1 and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set *f*(2) = 3. Next vertex is 5 so set *f*(3) = 5. In this fashion we get *f*(4) = 2, *f*(5) = 4. If we would continue, we would get back to *f*(1) =1 so this process is well defined and *f* is a morphism. Finally since *f* is bijective, *f* is an isomorphism.





Properties of Isomorphims

Since graphs are completely defined by their vertex sets and the number of edges between each pair, isomorphic graphs must have the same intrinsic properties. i.e. isomorphic graphs have the same…

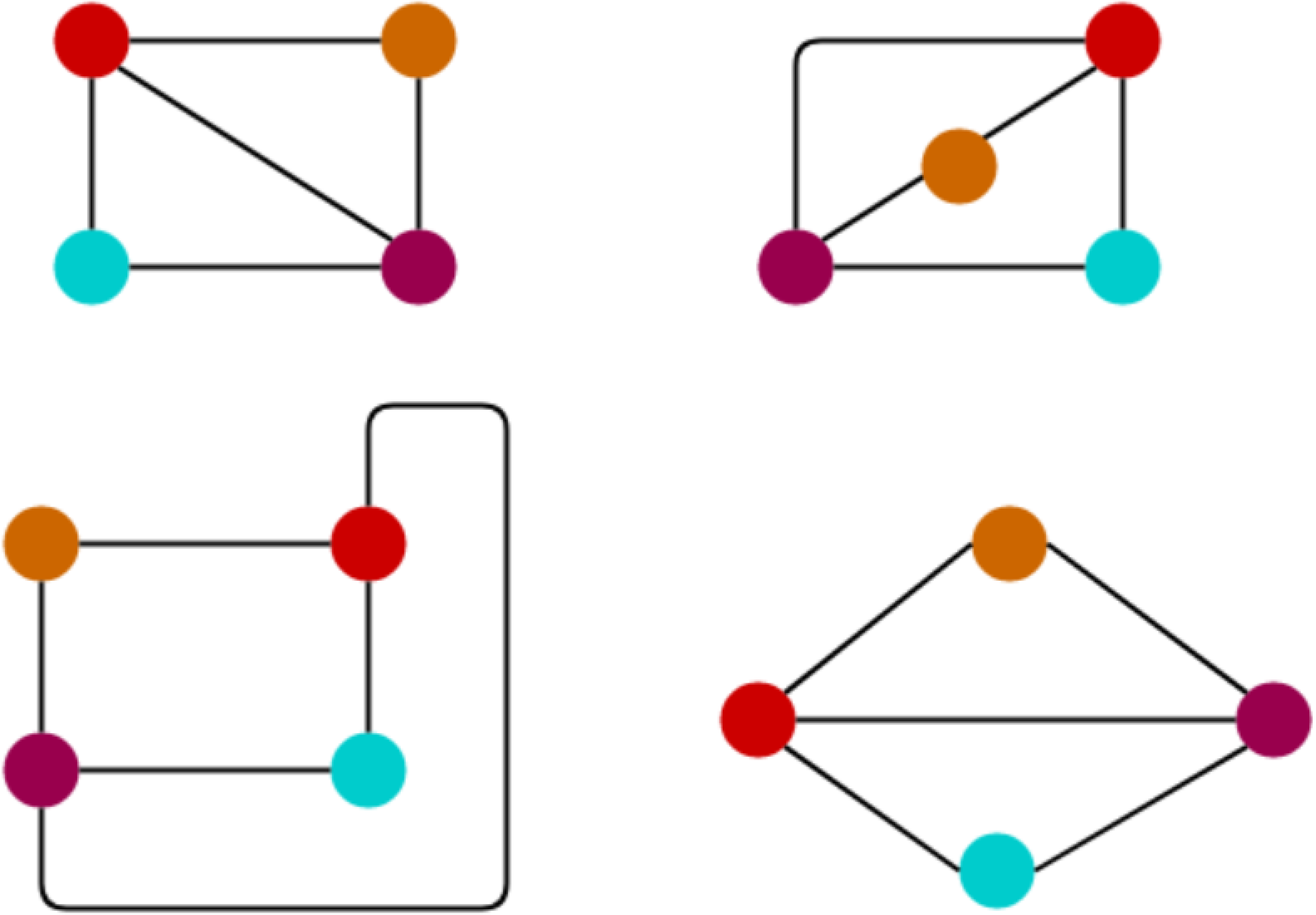
…number of vertices and edges

…degrees at corresponding vertices

…types of possible subgraphs

…any other property defined in terms of the basic graph theoretic building blocks!

Graph Isomorphism Example-



For any two graphs to be isomorphic, following 4 conditions must be satisfied-

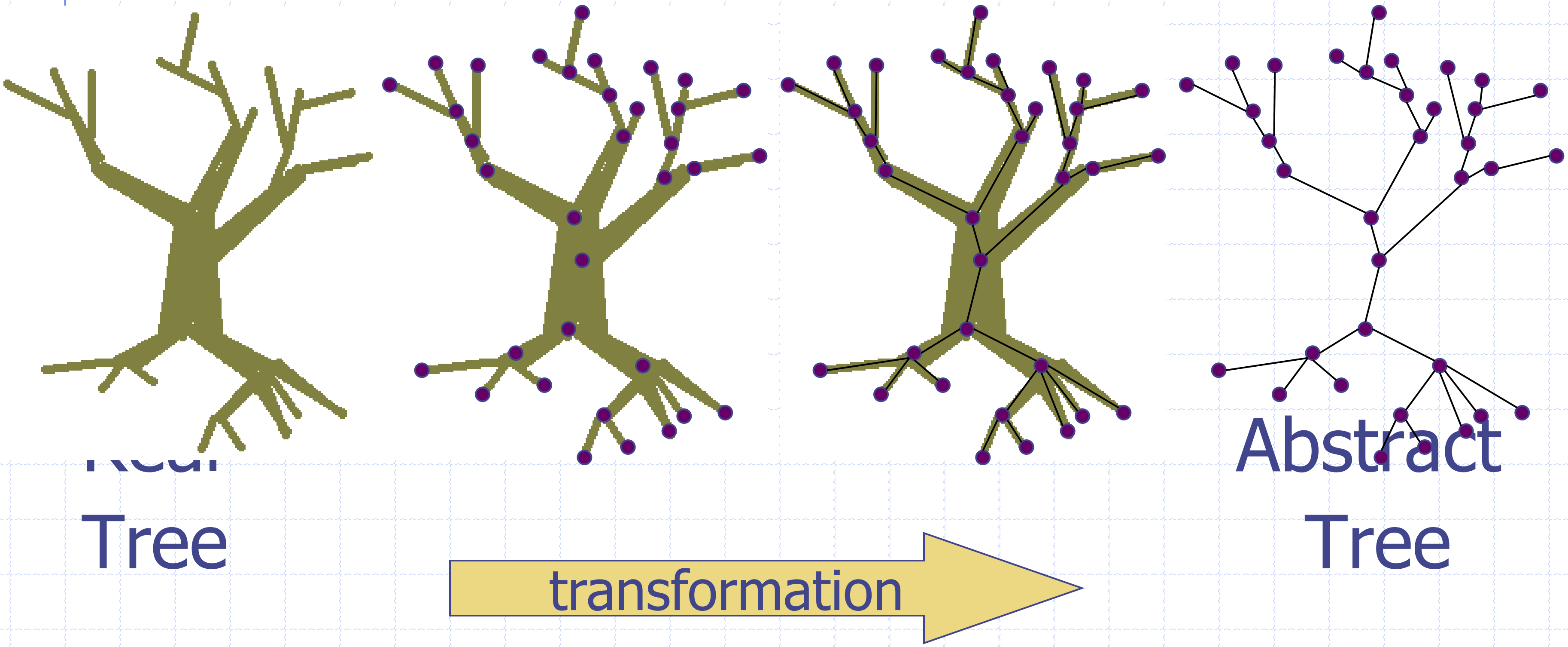
- Number of vertices in both the graphs must be same
- Number of edges in both the graphs must be same
- Degree sequence of both the graphs must be same
- If a cycle of length k is formed by the vertices

{ v_1 , v_2 , ….. , v_k } in one graph, then a cycle of same length k must be formed by the vertices { $f(v_1)$, $f(v_2)$, ….. , $f(v_k)$ } in the other graph as well

➔ *These conditions are necessary but not sufficient for any two graphs to be isomorphic*

Trees

Another type of graph in CS is called a *tree*:



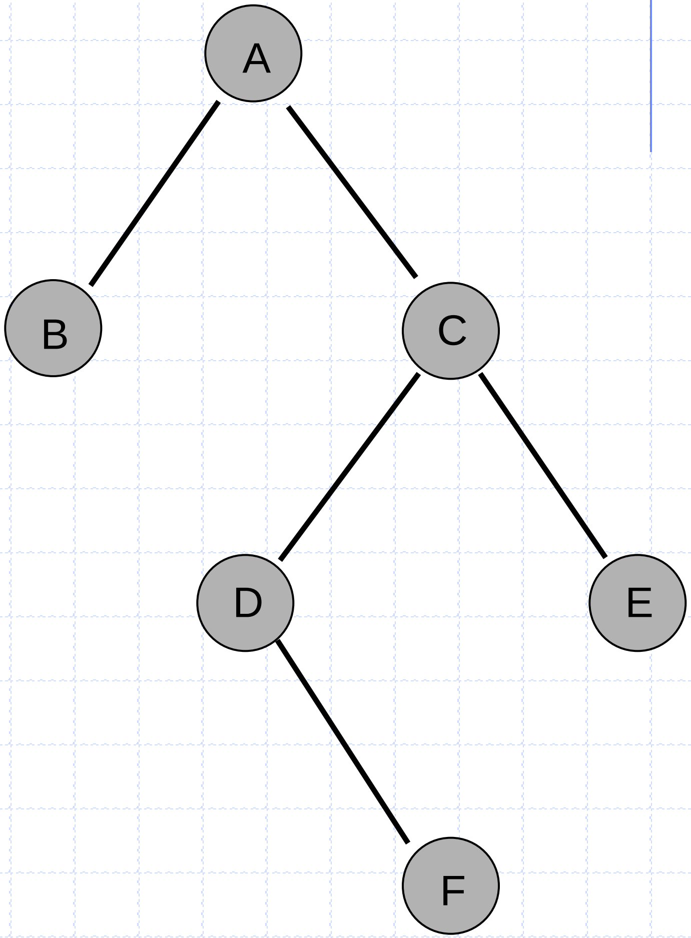
Real Tree

Abstract Tree

Trees

Let $G = (V, E)$ be an undirected graph. The following statements are equivalent,

1. G is a tree
2. Any two vertices in G are connected by unique simple path
3. G is connected, but if any edge is removed from E, the resulting graph is disconnected
4. G is connected, and $|E| = |V| - 1$
5. G is acyclic, for $|E| = |V| - 1$
6. G is acyclic, but if any edge is added to E, the resulting graph contains a cycle
7. A graph is tree if and only if it is minimally connected



Rank & Nullity

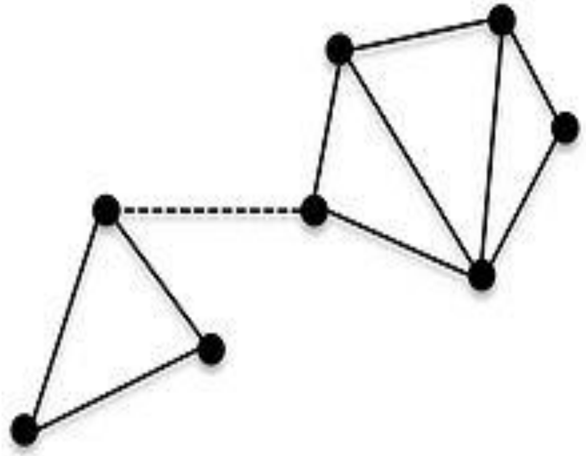
-For a graph G with n vertices, m edges & k components:

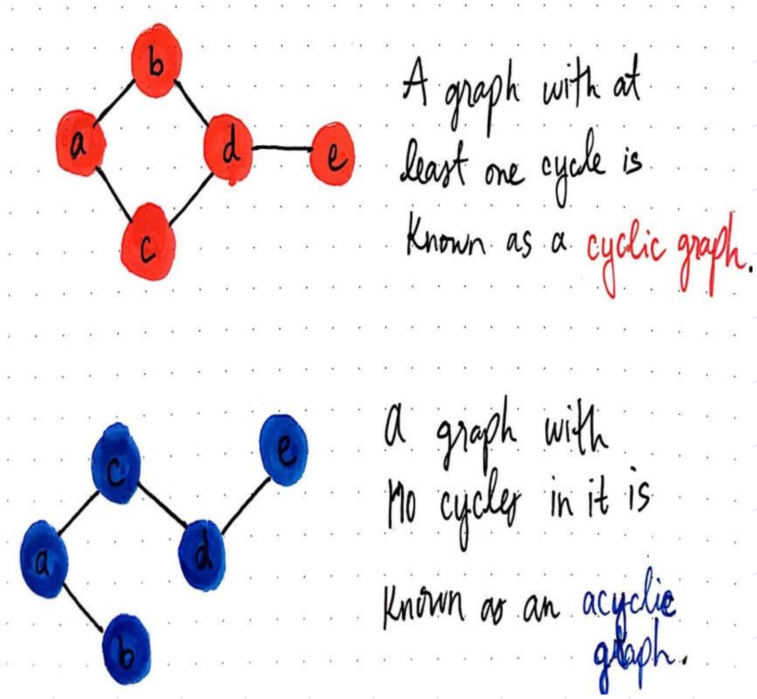
- Rank of G = n - k
- Nullity of G = m – Rank = m - n + k
- If G is connected;

Rank = n-1 and Nullity = m-n+1

- A graph G is **connected** if there is at least one path between every pair of vertices in G, Otherwise G is disconnected
- A graph is **cyclic** if the graph comprises a path that starts from a vertex and ends at the same vertex.
- A **finite** graph is with finite no. of edges and vertices.

Road network in a country ??




b
a
d
e
c
A graph with at least one cycle is known as a cyclic graph.
c
e
a
d
b
a graph with no cycles in it is known as an acyclic graph.

- A *path* is a sequence of vertices such that each vertex is adjacent to the next. In a path, each edge can be traveled **only once**.
- The *length of a path* is the number of edges in that path.
- A path that starts and ends at the same vertex is called a *circuit*.
- A graph is *connected* if any two vertices can be joined by a path. If this is not possible then the graph is *disconnected.*
- The connected parts of a disconnected graph are called *components.*
- A *bridge* is an edge in a connected graph whose removal makes it disconnected.
- A graph is said to be a planer if there exists some geometric representations of G which can be drawn on a plane such that no two of its edges intersect.
- An *Euler Path* is a path that travels through **every** edge of the graph **(once and only once).**
- An *Euler Circuit* is a circuit that travels through every edge of a G
- A *Walk* is a finite alternative sequence of vertices & edges.
- A circuit that contains each vertex in G exactly once, except for starting and ending vertex, which appears twice is known as *Hamiltonian circuit*.