CMSC 380

Graph Terminology and Representation

GRAPH BASICS

Basic Graph Definitions

- A ***graph*** G = (V,E) consists of a finite set of ***vertices***, V, and a finite set of ***edges***, E.
- Each edge is a pair (v,w) where v, w ∈ V.
  - V and E are sets, so each vertex v ∈ V is unique, and each edge e ∈ E is unique.
  - Edges are sometimes called ***arcs*** or ***lines***.
  - Vertices are sometimes called ***nodes*** or ***points***.

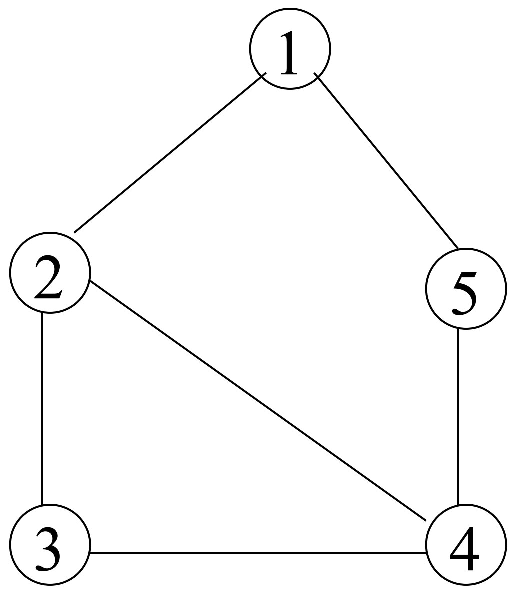
Graph Applications

- Graphs can be used to model a wide range of applications including
- Intersections and streets within a city
- Roads/trains/airline routes connecting cities/ countries
- Computer networks
- Electronic circuits

Basic Graph Definitions (2)

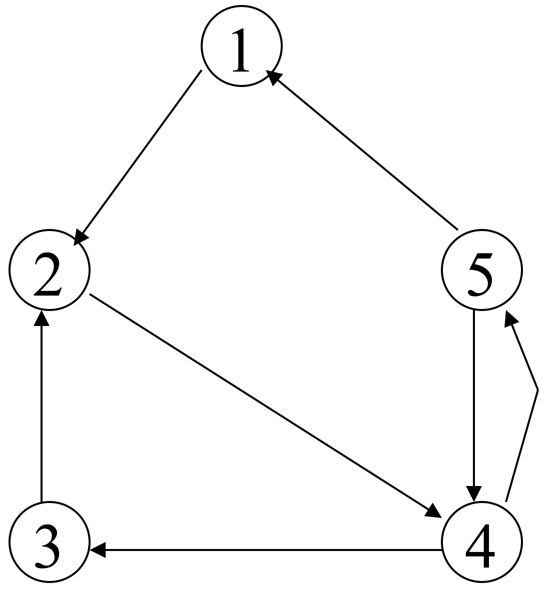
- A ***directed graph*** is a graph in which the edges are ordered pairs.
  That is, $(u,v) \neq (v,u)$, $u, v \in V$.
  Directed graphs are sometimes called ***digraphs***.
- An ***undirected graph*** is a graph in which the edges are unordered pairs.
  That is, $(u,v) = (v,u)$.
- A ***sparse graph*** is one with "few" edges.
  That is $|E| = O(|V|)$
- A ***dense graph*** is one with "many" edges.
  That is $|E| = O(|V|^2)$

Undirected Graph



- All edges are two-way. Edges are unordered pairs.
- V = { 1, 2 ,3, 4, 5}
- E = { (1,2), (2, 3), (3, 4), (2, 4), (4, 5), (5, 1) }

Directed Graph



- All edges are “one-way” as indicated by the arrows. Edges are ordered pairs.
- V = { 1, 2, 3, 4, 5}
- E = { (1, 2), (2, 4), (3, 2), (4, 3), (4, 5), (5, 4), (5, 1) }

Number of Edges in a Graph

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

Number of Edges in a Graph

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

A: For undirected graphs, the number of pairs in *V:*

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

For directed graphs, the number of ordered-pairs in *V:*

$$= n^2 - n = n \cdot (n-1)$$

Number of Possible Graphs

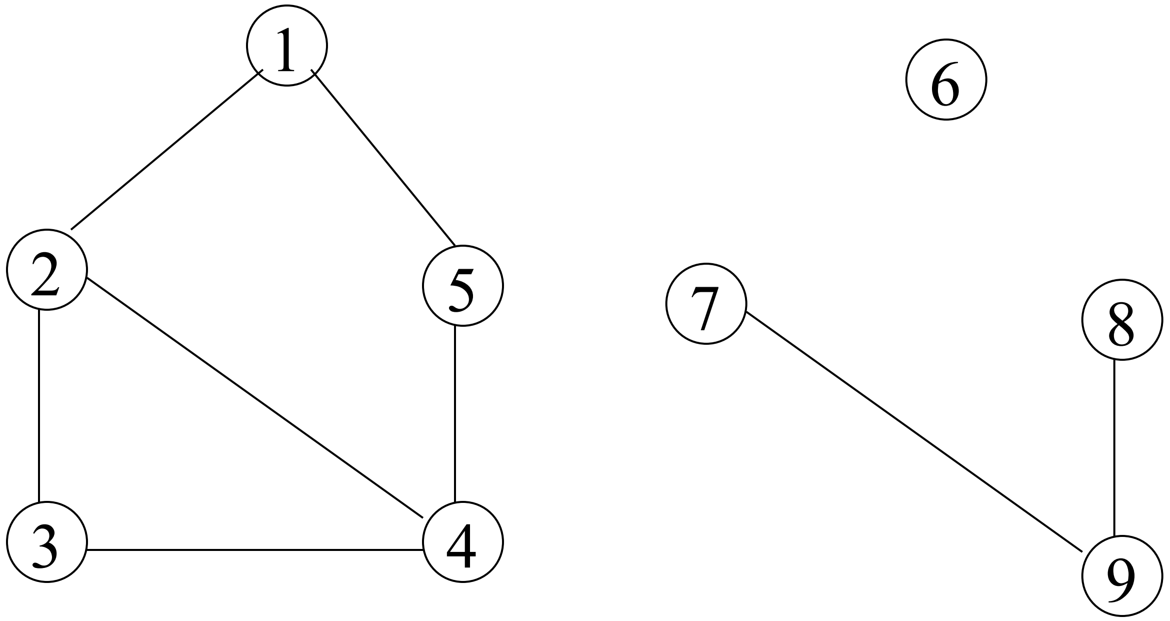
Q: How many possible simple undirected graphs are there for the same set of vertices *V* ?

Number of Possible Graphs

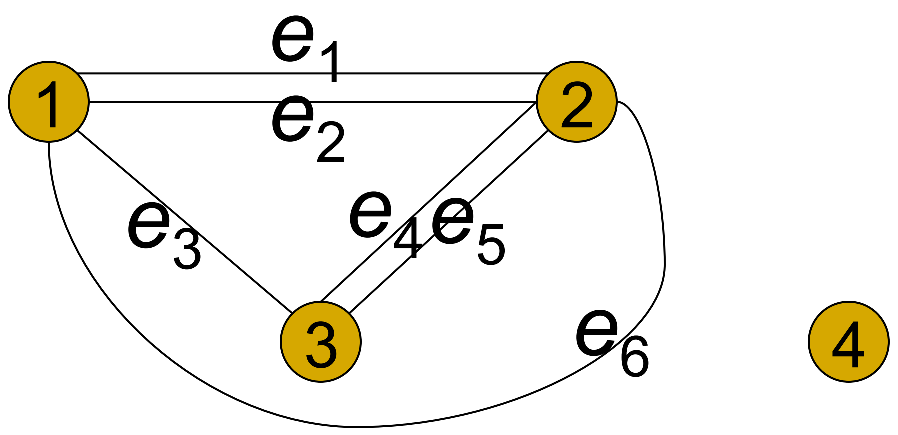
Q: How many possible simple undirected graphs are there for the same set of vertices V ?

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 $2^{n(n-1)/2}$

A Single Graph with Multiple Components



Multigraphs



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

$e_1 \rightarrow \{1,2\}$, $e_2 \rightarrow \{1,2\}$, $e_3 \rightarrow \{1,3\}$,
$e_4 \rightarrow \{2,3\}$, $e_5 \rightarrow \{2,3\}$, $e_6 \rightarrow \{1,2\}$

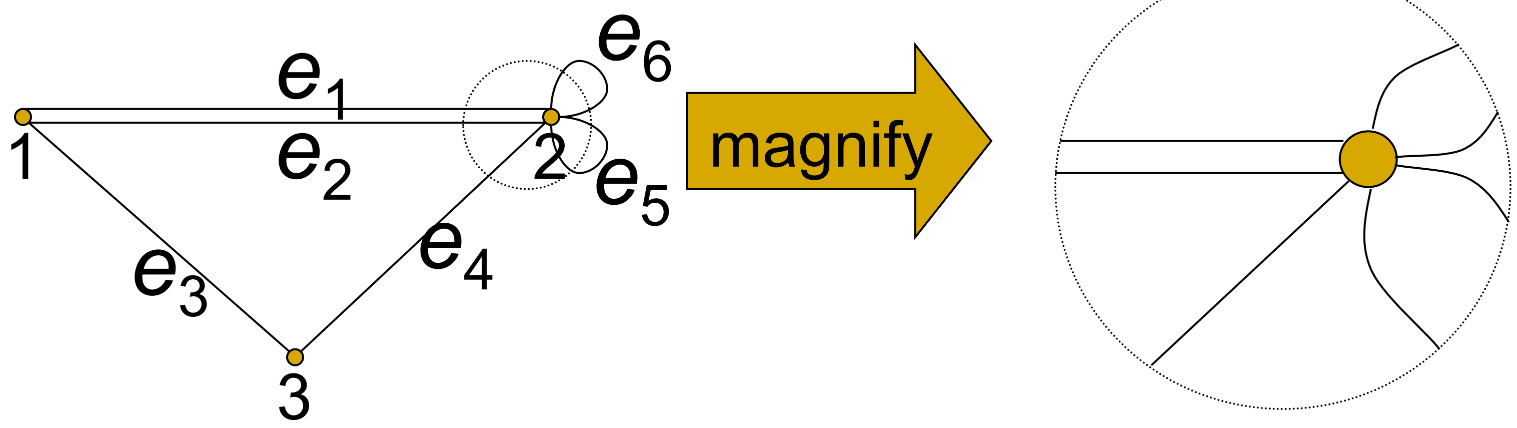
Basic Graph Definitions (3)

- Vertex w is ***<u>adjacent to</u>*** vertex v if and only if (v, w) ∈ E.
- For undirected graphs, with edge (v, w), and hence also (w, v), w is adjacent to v and v is adjacent to w.
- An edge may also have:
  - ***weight*** or ***cost*** -- an associated value
  - ***label*** -- a unique name
- We can define edge properties via a function

  f : E → Y, where Y is the set of values for that property

Degree

The ***degree*** of a vertex counts the number of edges that *seem* to be sticking out if you looked under a magnifying glass:

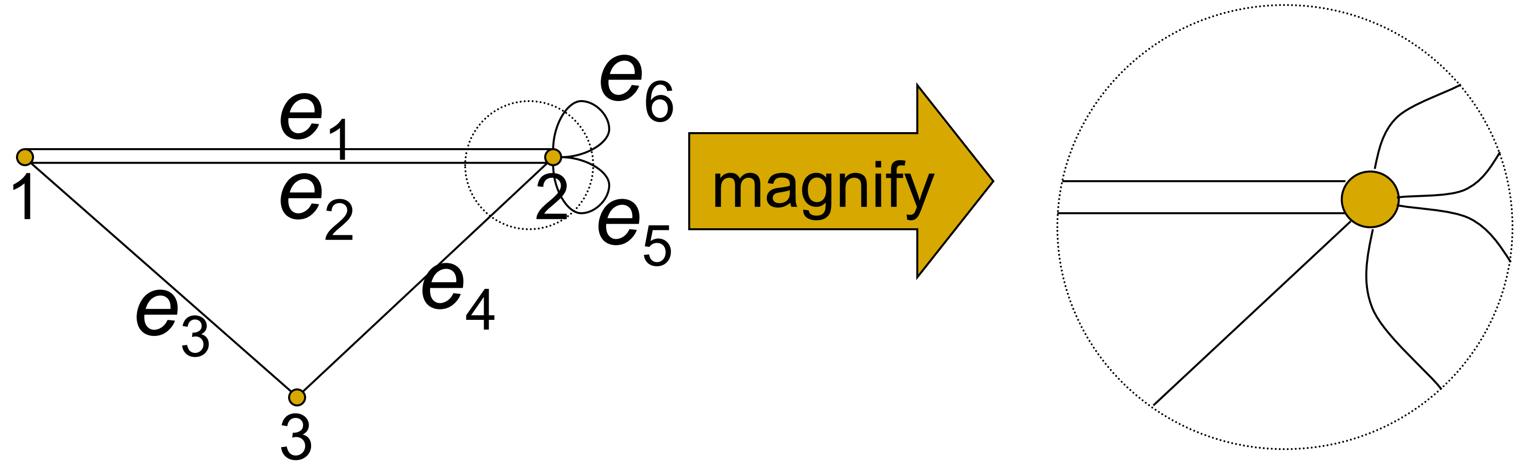


Thus deg(2) = 7 even though the vertex is only incident with 5 edges.

Q: How to define this formally?

Degree

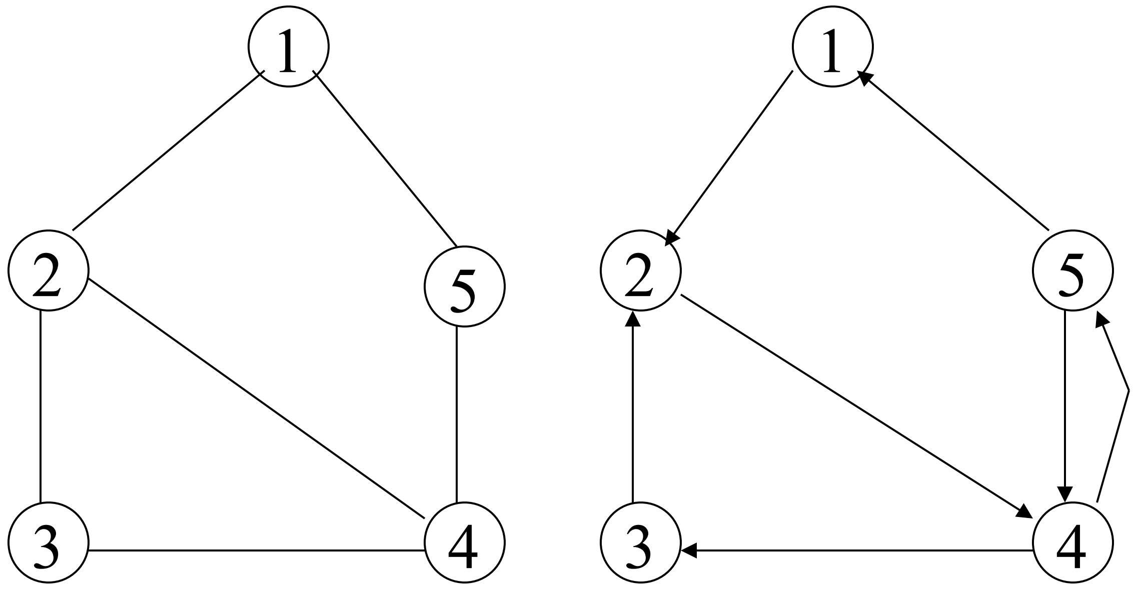
A: Add 1 for every regular edge incident with the vertex and 2 for every loop. Thus deg(2) = 1 + 1 + 1 + 2 + 2 = 7



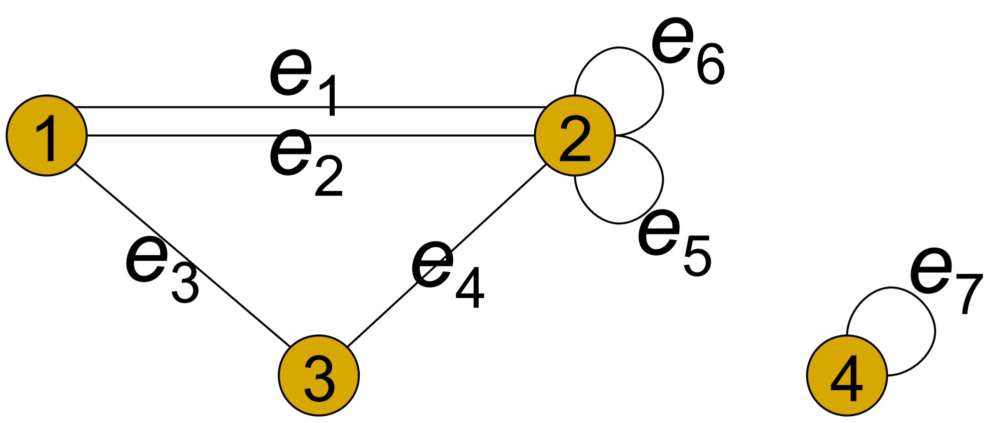
Degree is sometimes called *valence.*

Degree for Directed Graphs

- For directed graphs vertex w is ***adjacent to*** vertex v if and only if (v, w) ∈ E.
- ***Indegree*** $deg^-(w)$ is the number of edges (v,w).
- ***OutDegree*** $deg^+(w)$ is the number of edges(w,v).



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

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

Q: In a party of 5 people can each person be friends with exactly three others?

Handshaking Theorem

A: Imagine a simple graph with 5 people as *vertices* and edges being undirected edges between friends (simple graph assuming friendship is symmetric and irreflexive). Number of friends each person has is the degree of the person.

Handshaking would imply that

$|E| = (\text{sum of degrees})/2$ or

$2|E| = (\text{sum of degrees}) = (5\cdot 3) = 15.$

Impossible as 15 is not even. In general:

Handshaking Theorem

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

Proof (by contradiction): Assume that this is false. Then, we'd have

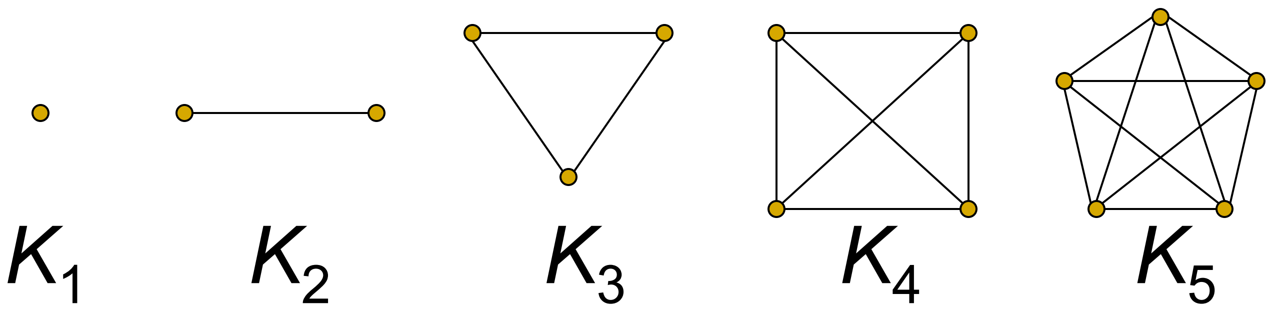
$$\sum_{e \in E} \deg(e) = 2|E| = \sum_{e \in E\ s.t.\ \mathrm{even}(\deg(e))} deg(e) + \sum_{e \in E\ s.t.\ \mathrm{odd}(\deg(e))} deg(e)$$

The second term must be odd (since the sum of an odd number of odds is odd), and so we have 2E = even number + odd number. This is impossible.

GRAPH PATTERNS

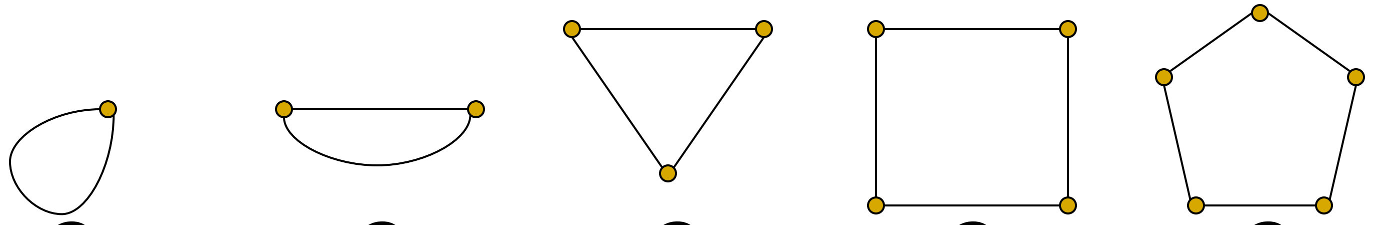
Graph Patterns: Complete Graphs - K_n

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.



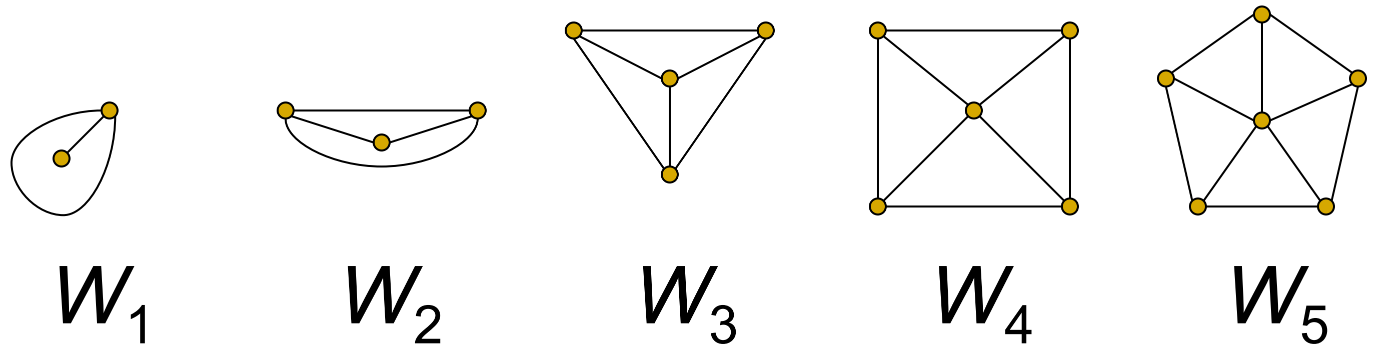
Graph Patterns: Cycles - C_n

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



Graph Patterns: Wheels - W_n

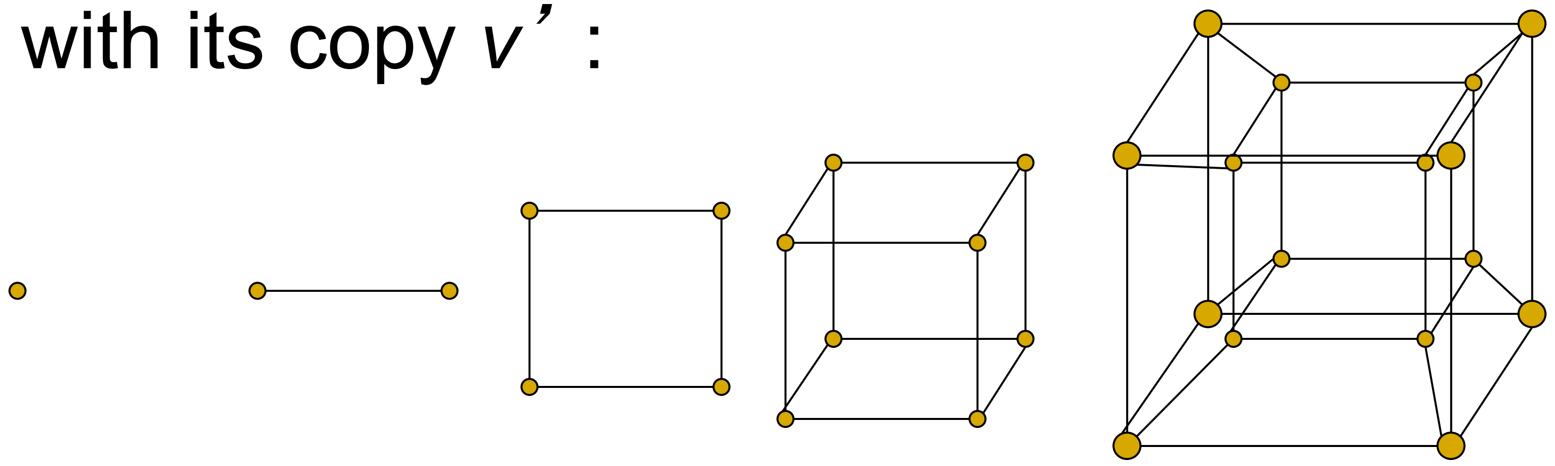
The ***wheel graph*** W_n is just a cycle graph with an extra vertex in the middle:



Usually consider wheels with 3 or more spokes only.

Graph Patterns: Cubes - Q_n

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



Q_0 Q_1 Q_2 Q_3 Q_4 (hypercube)

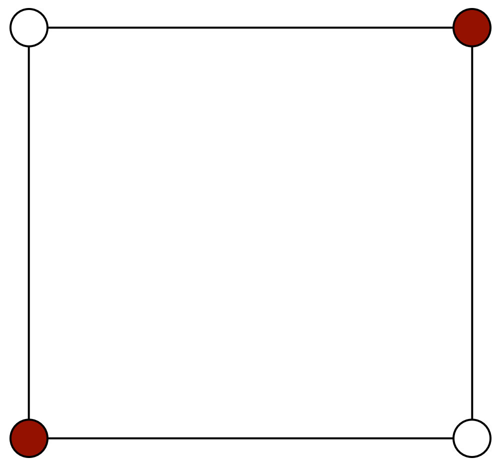
Bipartite Graphs

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.

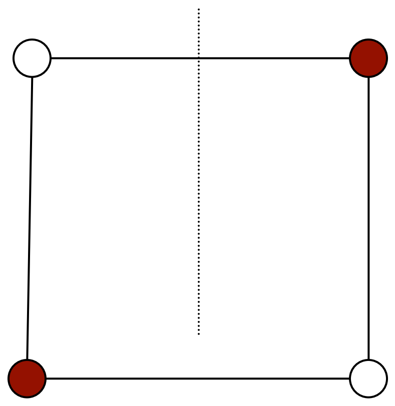
Another way of expressing the same idea is ***bichromatic*** : vertices can be colored using two colors so that no two vertices of the same color are adjacent.

Bipartite Graphs

EG: C_4 is a bichromatic:

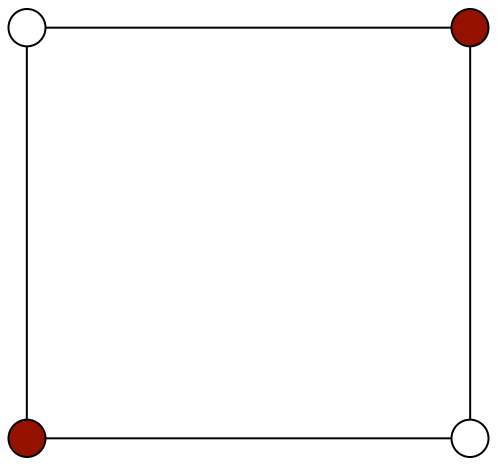


And so is bipartite, if we redraw it:

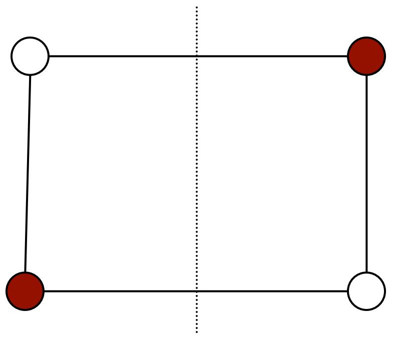


Bipartite Graphs

EG: C_4 is a bichromatic:

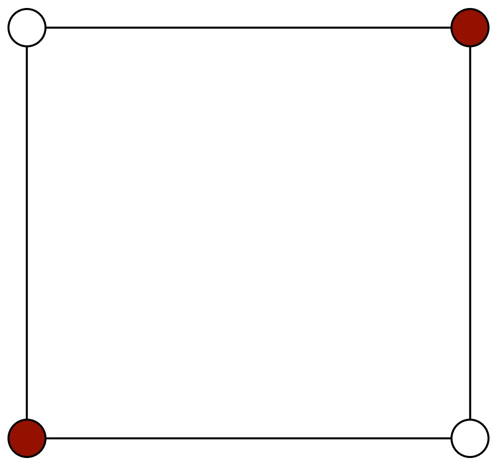


And so is bipartite, if we redraw it:

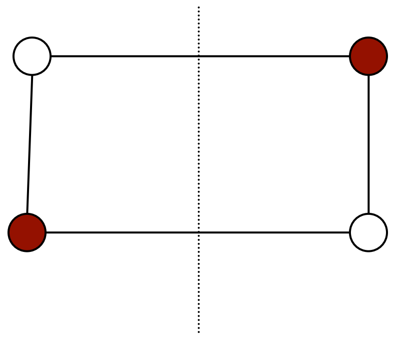


Bipartite Graphs

EG: C_4 is a bichromatic:

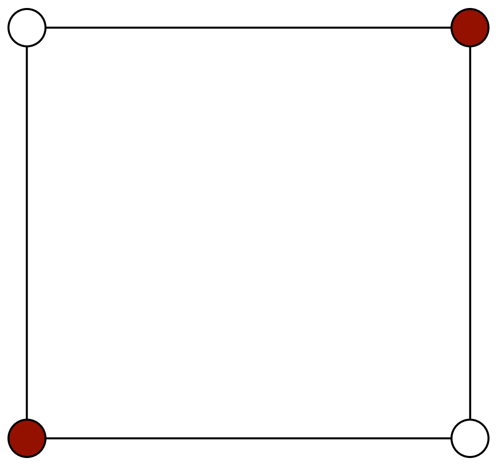


And so is bipartite, if we redraw it:

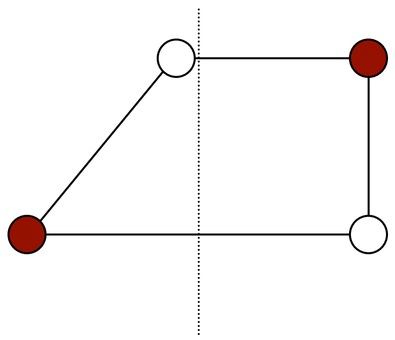


Bipartite Graphs

EG: C_4 is a bichromatic:

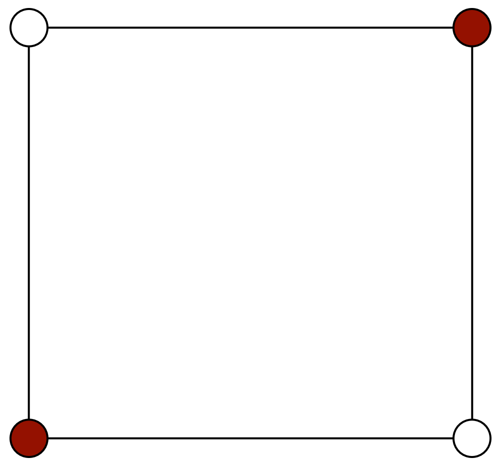


And so is bipartite, if we redraw it:

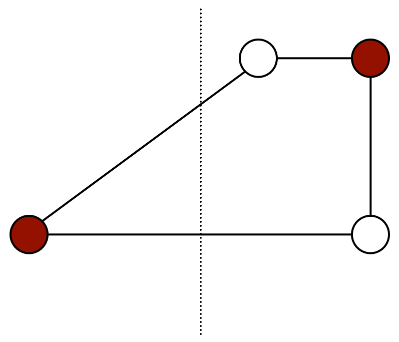


Bipartite Graphs

EG: C_4 is a bichromatic:

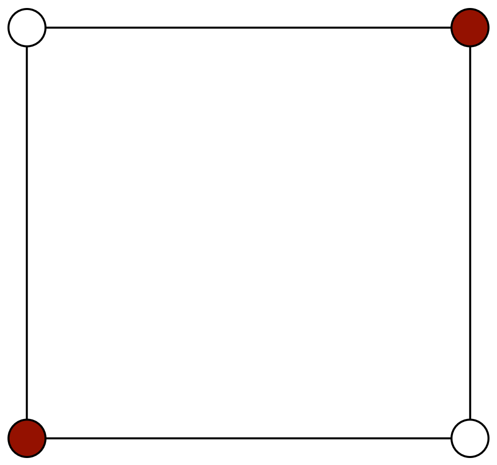


And so is bipartite, if we redraw it:

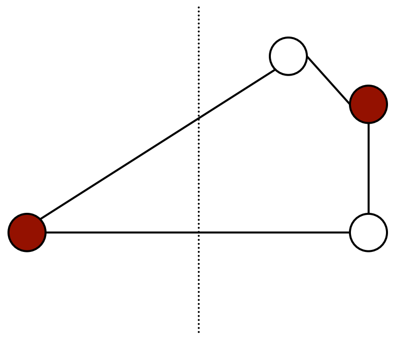


Bipartite Graphs

EG: C_4 is a bichromatic:

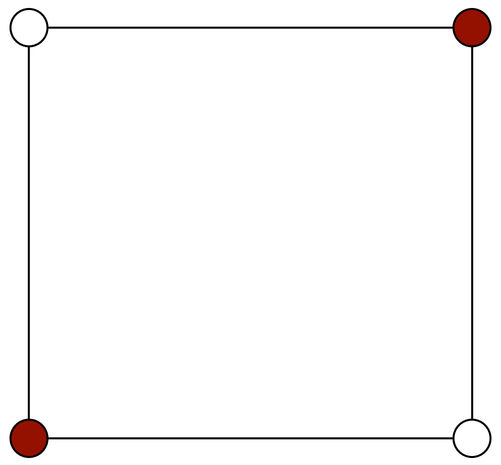


And so is bipartite, if we redraw it:

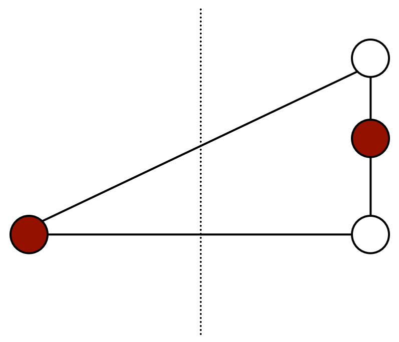


Bipartite Graphs

EG: C_4 is a bichromatic:

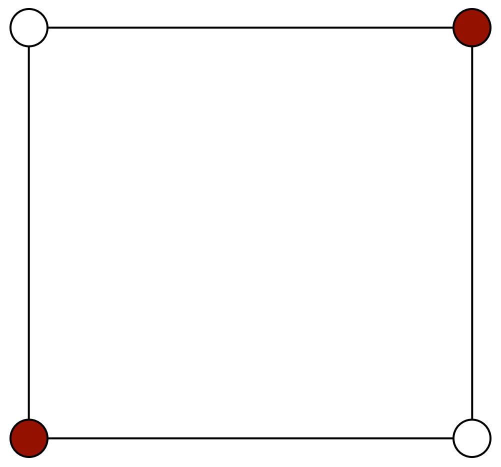


And so is bipartite, if we redraw it:

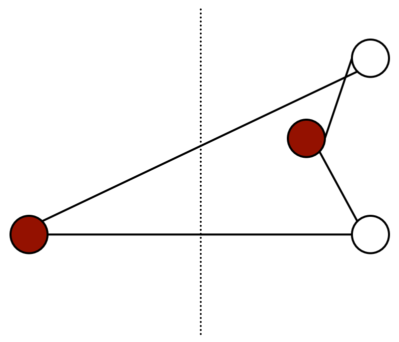


Bipartite Graphs

EG: C_4 is a bichromatic:

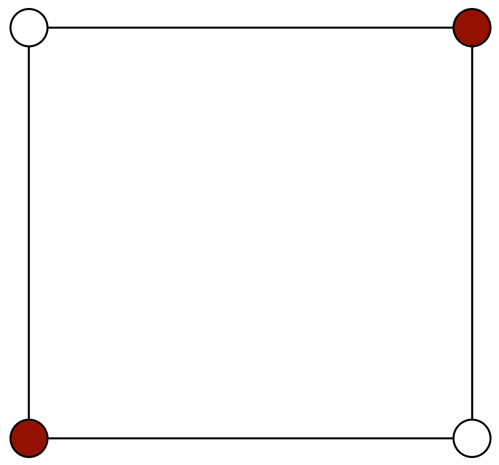


And so is bipartite, if we redraw it:

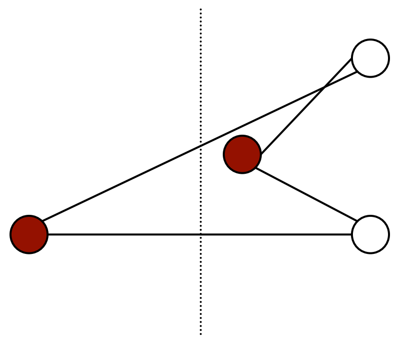


Bipartite Graphs

EG: C_4 is a bichromatic:

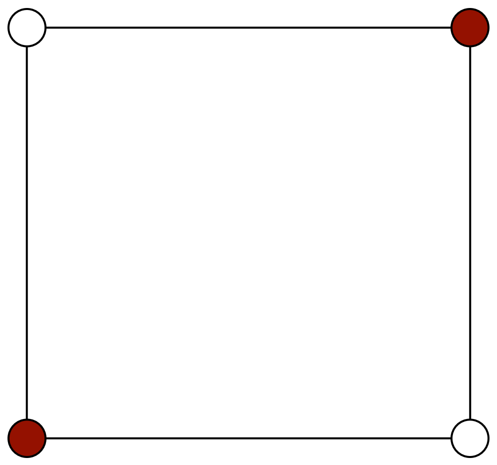


And so is bipartite, if we redraw it:

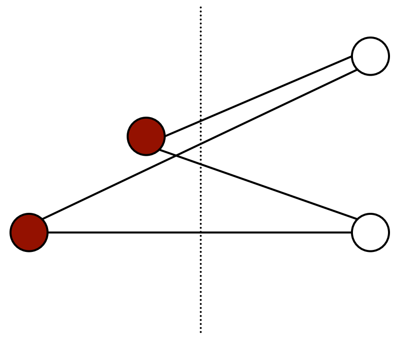


Bipartite Graphs

EG: C_4 is a bichromatic:

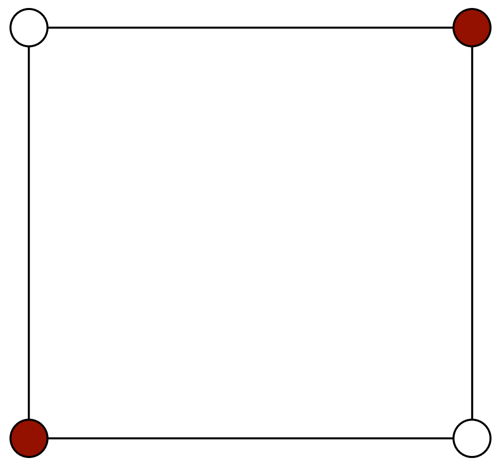


And so is bipartite, if we redraw it:

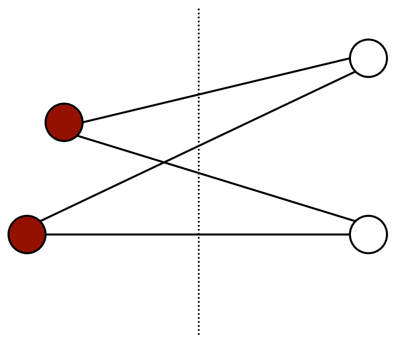


Bipartite Graphs

EG: C_4 is a bichromatic:

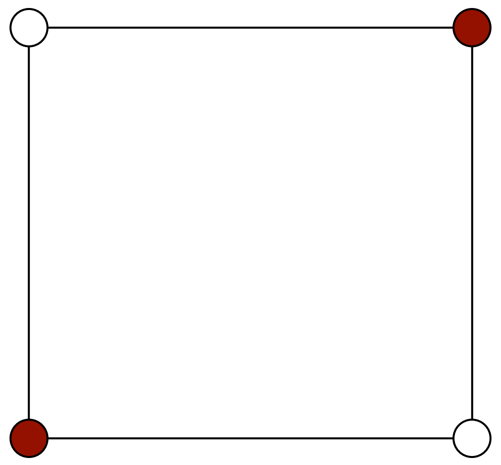


And so is bipartite, if we redraw it:

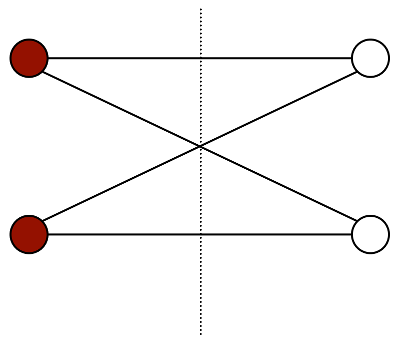


Bipartite Graphs

EG: C_4 is a bichromatic:



And so is bipartite, if we redraw it:



Bipartite Graphs

Q: For which n is C_n bipartite?

Bipartite Graphs

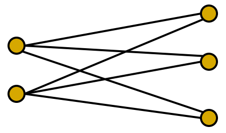
Q: For which *n* is C_n bipartite?

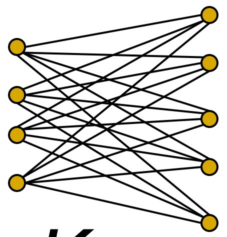
A: C_n is bipartite when *n* is even. For even *n* color all odd numbers red and all even numbers green so that vertices are only adjacent to opposite color.

If *n* is odd, C_n is not bipartite. If it were, color 0 red. So 1 must be green, and 2 must be red. This way, all even numbers must be red, including vertex *n*-1. But *n*-1 connects to 0 $\rightarrow\leftarrow$.

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. For example,


$K_{2,3}$


$K_{4,5}$

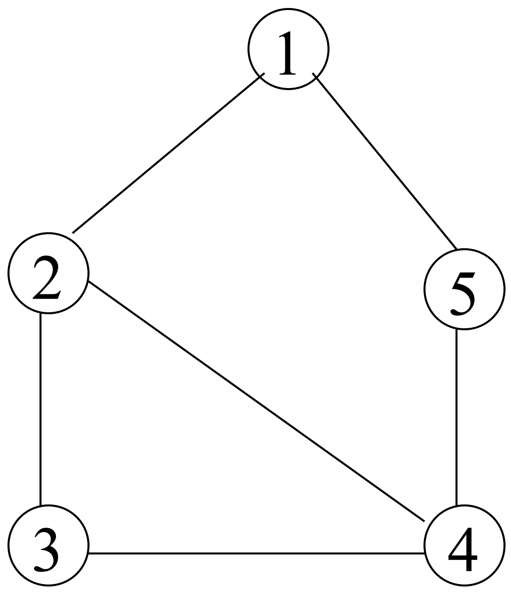
PATHS AND COMPONENTS

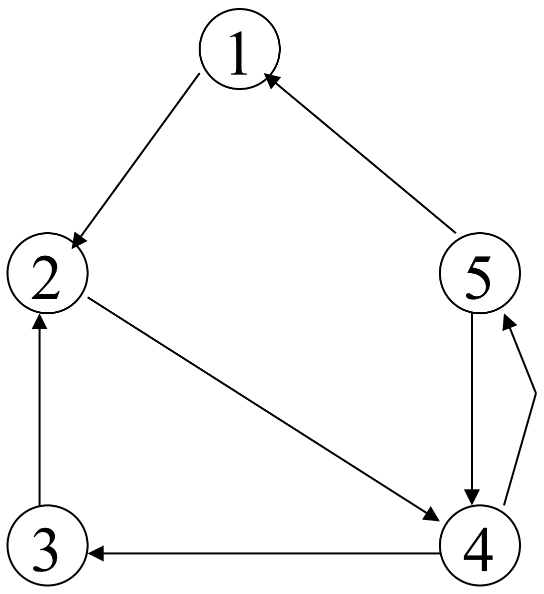
Paths in Graphs

- A ***path*** in a graph is a sequence of vertices $w_1, w_2, w_3, \ldots, w_n$ such that $(w_i, w_{i+1}) \in E$ for $1 \leq i < n$.
- The ***length*** of a path in a graph is the number of edges on the path. The length of the path from a vertex to itself is 0.
- A ***simple path*** is a path such that all vertices are distinct, except that the first and last may be the same.
- A ***cycle*** in a graph is a path $w_1, w_2, w_3, \ldots, w_n$, $w \in V$ such that:
  - there are at least two vertices on the path
  - $w_1 = w_n$ (the path starts and ends on the same vertex)
  - if any part of the path contains the subpath w_i, w_j, w_i, then each of the edges in the subpath is distinct (i. e., no backtracking along the same edge)
- A ***simple cycle*** is one in which the path is simple.
- A directed graph with no cycles is called a ***directed acyclic graph***, often abbreviated as DAG

Paths in Graphs (2)

- How many simple paths from 1 to 4 and what are their lengths?





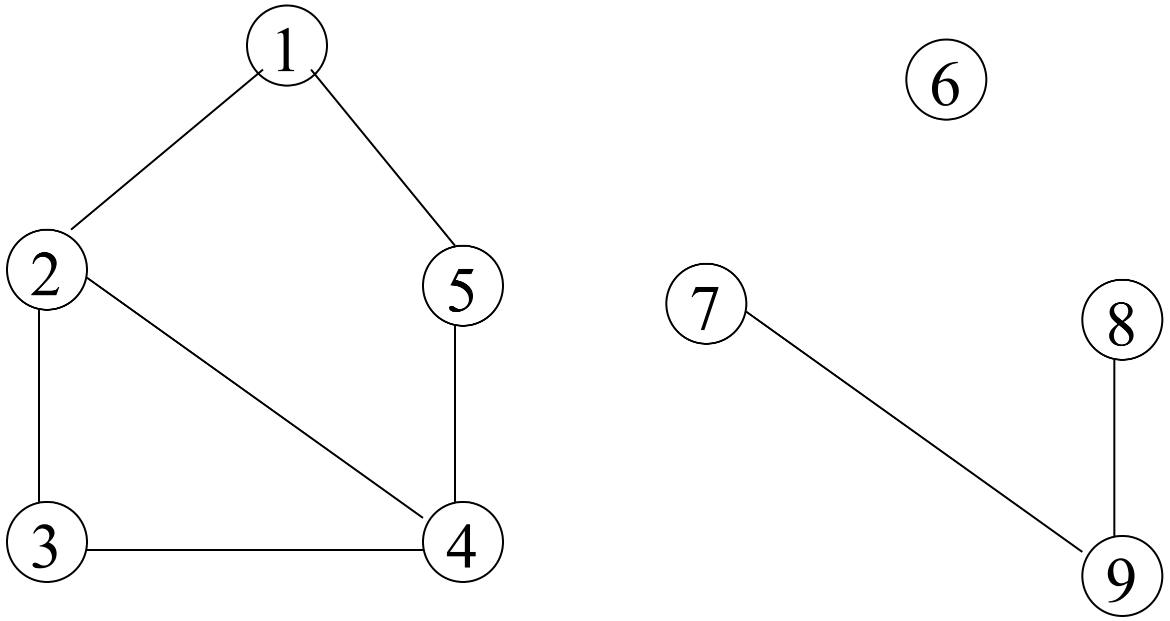
Connectedness in Graphs

- An undirected graph is ***connected*** if there is a path from every vertex to every other vertex.
- A directed graph is ***strongly connected*** if there is a path from every vertex to every other vertex.
- A directed graph is ***weakly connected*** if there would be a path from every vertex to every other vertex, disregarding the direction of the edges.
- A ***connected component*** of a graph is any maximal connected subgraph. Connected components are sometimes simply called ***components***.

Disjoint Sets and Graphs

- Disjoint sets can be used to determine connected components of an undirected graph.
- For each edge, place its two vertices (u and v) in the same set -- i.e. union(u, v)
- When all edges have been examined, the forest of sets will represent the connected components.
- Two vertices, x, y, are connected if and only if find(x) = find(y)

Undirected Graph/Disjoint Set Example



Sets representing connected components

{ 1, 2, 3, 4, 5 }

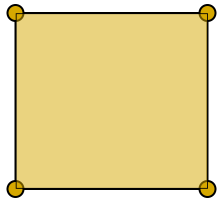
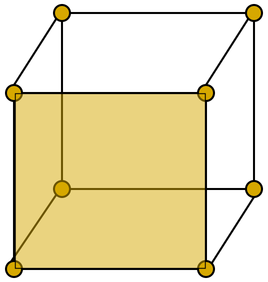
{ 6 }

{ 7, 8, 9 }

DiGraph / Strongly Connected Components

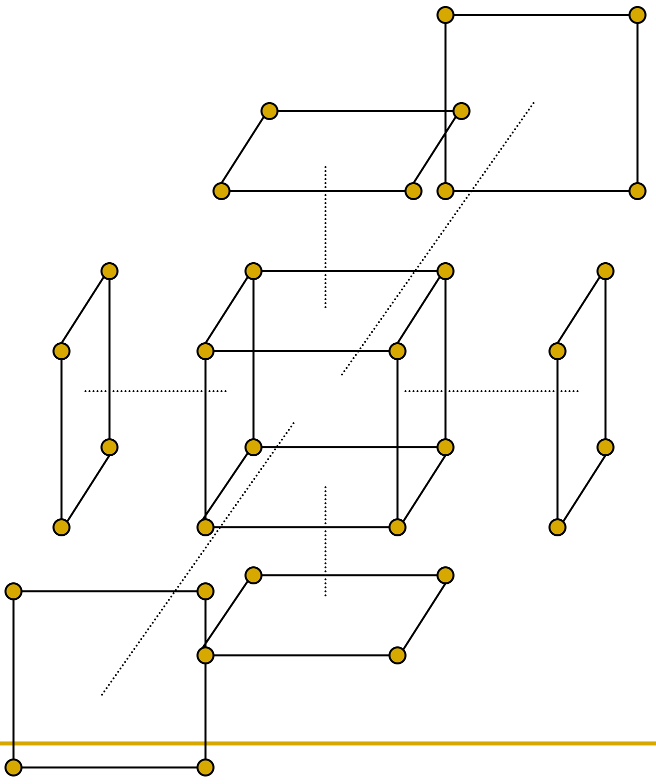
$$\begin{tikzcd}
a \arrow[dr] & b \arrow[l] & g & \\
d \arrow[u] & c \arrow[u] \arrow[r] & f \arrow[ul] \arrow[u] \arrow[r] & h \arrow[ul] \arrow[dl] \\
e \arrow[u] \arrow[ur] & j \arrow[urr] & i \arrow[l] &
\end{tikzcd}$$

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?

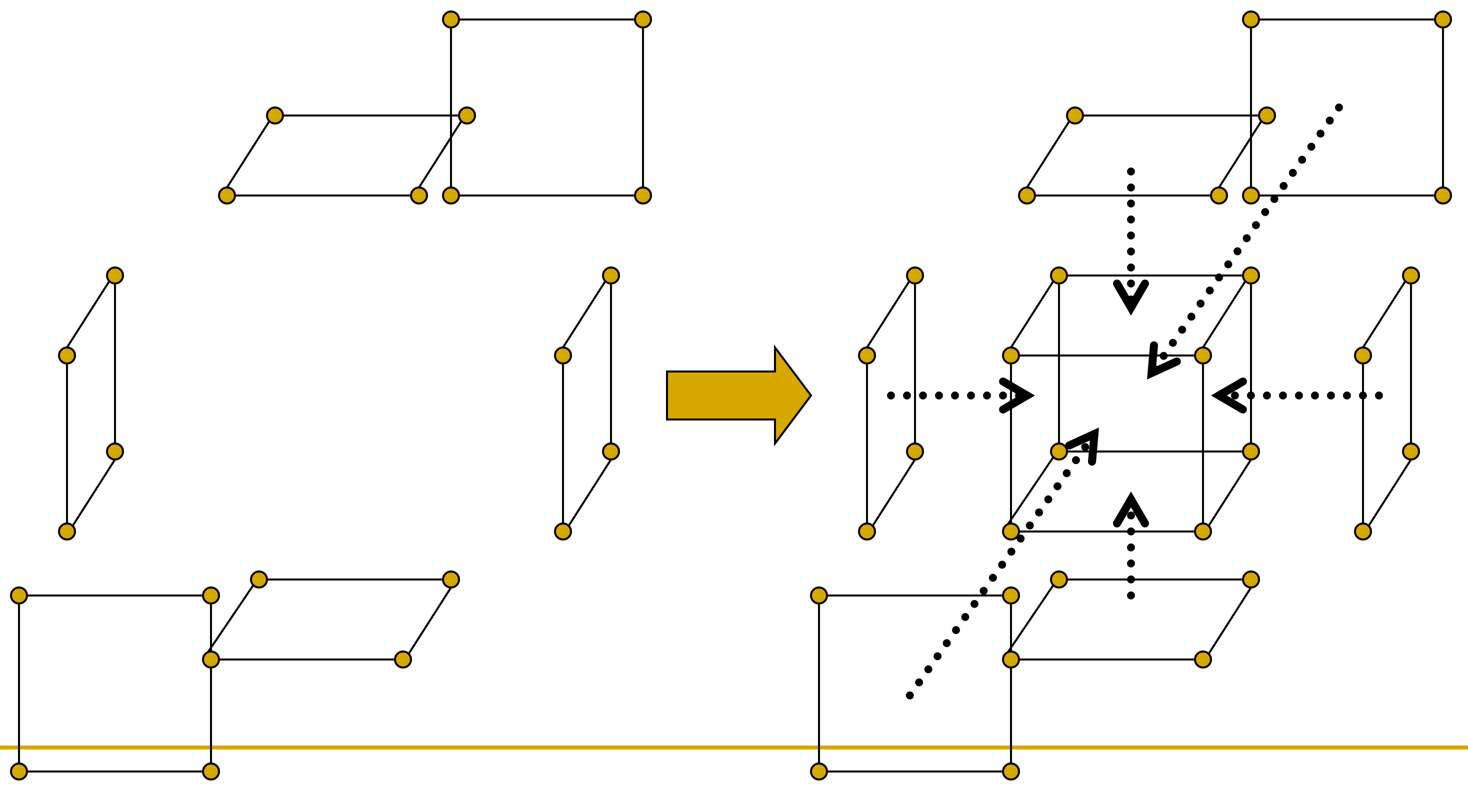
Subgraphs

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:



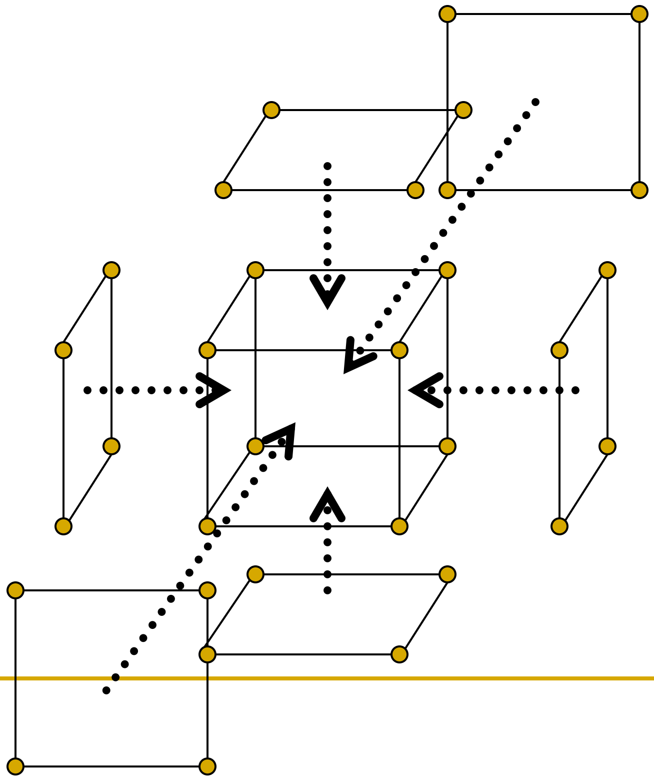
Unions

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

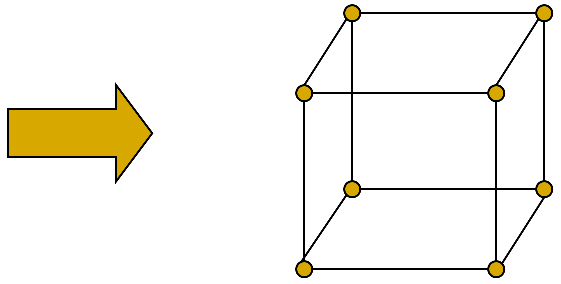


Unions

If we assign the 2-cube faces (aka *S*quares) 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. That is, $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$.

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

GRAPH REPRESENTATION

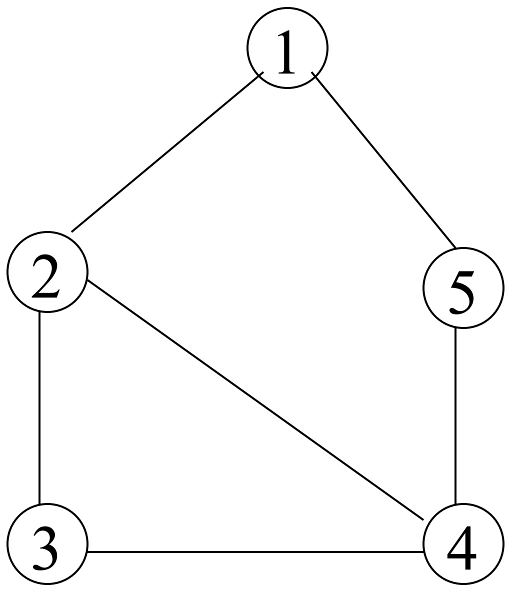
Graph Representation

- Data elements:
  - Vertices and Edges
- Operations:
  - getDegree(u) -- Returns the degree of vertex u (outdegree of vertex u in directed graph)
  - getAdjacent(u) -- Returns a list of the vertices **adjacent to** vertex u (list of vertices that u points to for a directed graph)
  - isAdjacentTo(u, v) -- Returns TRUE if vertex v is adjacent to vertex u, FALSE otherwise.
- Has some associated algorithms to be discussed.

Adjacency Matrix Implementation

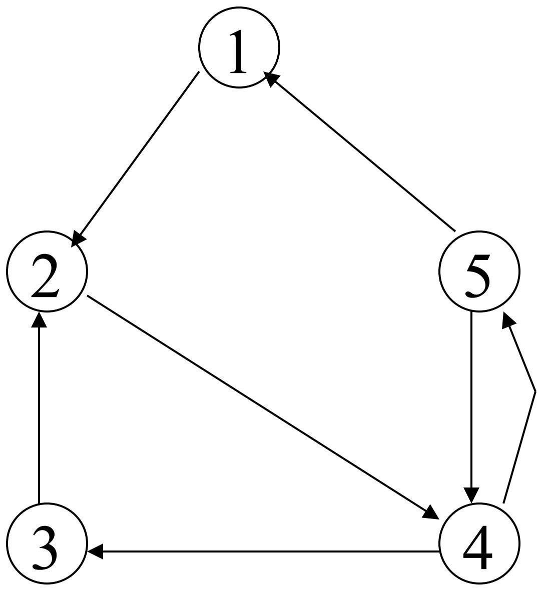
- Uses array of size $|V| \times |V|$ where each entry (i ,j) is boolean
  - TRUE if there is an edge from vertex i to vertex j
  - FALSE otherwise
  - store weights when edges are weighted
- Very simple, but large space requirement = $O(|V|^2)$
- Appropriate if the graph is dense.
- Otherwise, most of the entries in the table are FALSE.
- For example, if a graph is used to represent a street map like Manhattan in which most streets run E/W or N/S, each intersection is attached to only 4 streets and $|E| < 4*|V|$. If there are 3000 intersections, the table has 9,000,000 entries of which only 12,000 are TRUE.

Undirected Graph / Adjacency Matrix



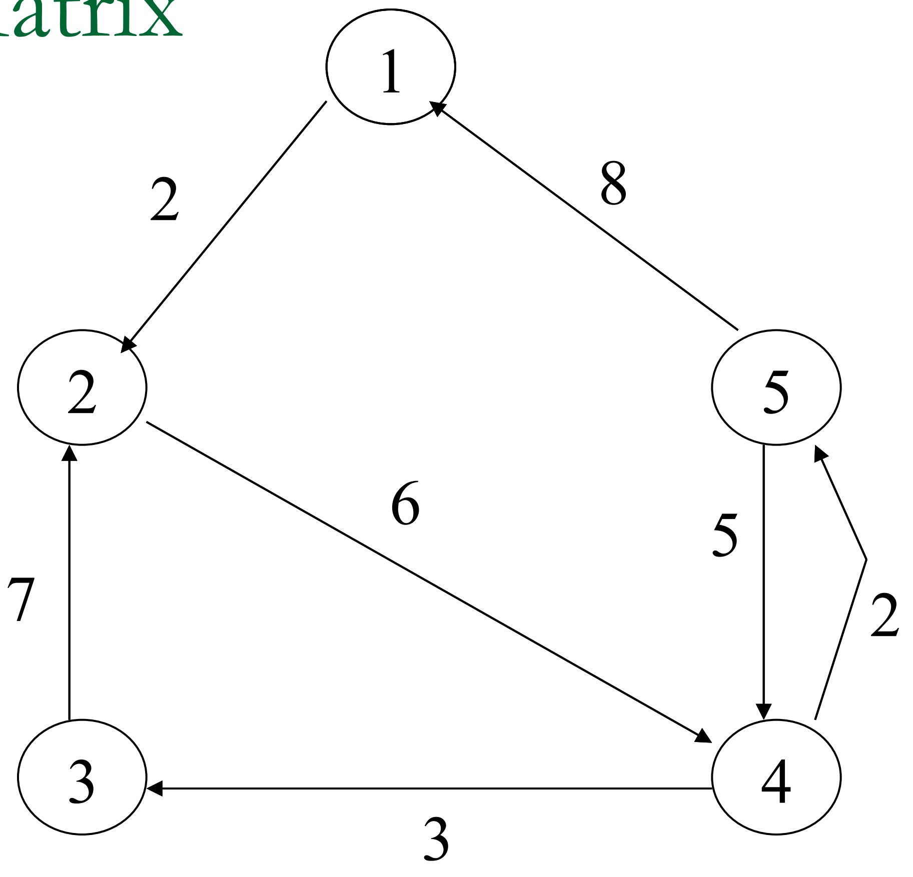
	1	**2**	**3**	**4**	**5**
1	0	1	0	0	1
2	1	0	1	1	0
3	0	1	0	1	0
4	0	1	1	0	1
5	1	0	0	1	0

Directed Graph / Adjacency Matrix



	1	**2**	**3**	**4**	**5**
1	0	1	0	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	1	0	0	1	0

Weighted, Directed Graph / Adjacency Matrix



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

Adjacency Matrix Performance

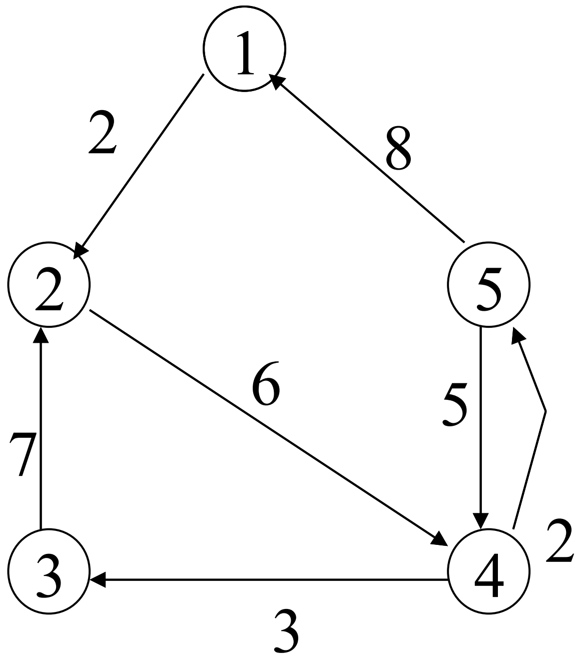
- Storage requirement: O($|V|^2$)
- Performance:

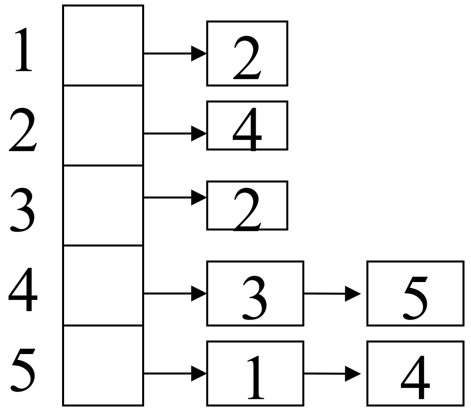
getDegree (u)	
isAdjacentTo(u, v)	
getAdjacent(u)	

Adjacency List Implementation

- If the graph is sparse, then keeping a list of adjacent vertices for each vertex saves space. Adjacency Lists are the commonly used representation. The lists may be stored in a data structure or in the Vertex object itself.
  - **Vector of lists**: A vector of lists of vertices. The i-th element of the vector is a list, $L_{i,}$ of the vertices adjacent to v_i.
- If the graph is sparse, then the space requirement is O(|E| + |V|), "linear in the size of the graph"
- If the graph is dense, then the space requirement is O($|V|^2$)

Vector of Lists





Adjacency List Performance

- Storage requirement:
- Performance:

getDegree(u)	
isAdjacentTo(u, v)	
getAdjacent(u)	

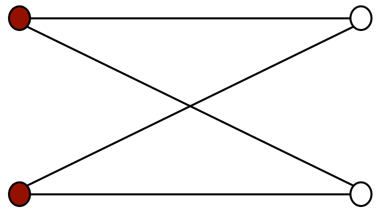
GRAPH ISOMORPHISM

Graph Isomorphism

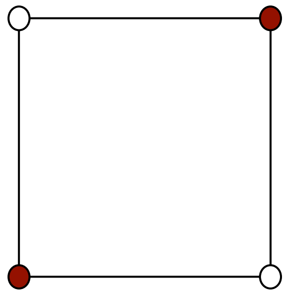
Intuitively, two graphs are isomorphic if can bend, stretch and reposition vertices of the first graph, until the second graph is formed.

Etymologically, *isomorphic* means "same shape".

EG: Can twist or relabel:



to obtain:



Graph Isomorphism

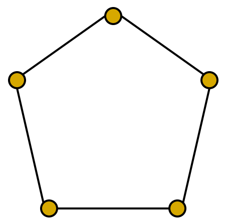
DEF: Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are undirected or directed multigraphs.

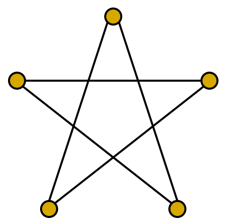
Let $f : V_1 \rightarrow V_2$ be a function s.t.:

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 from $f(u)$ to $f(v)$ in G_2.

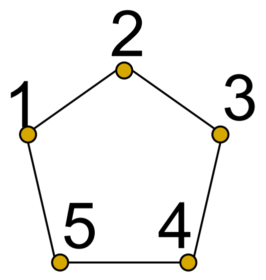
Then f is called an ***isomorphism*** and G_1 is said to be ***isomorphic*** to G_2.

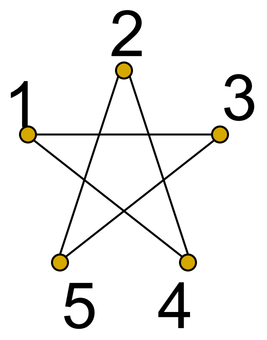
Graph Isomorphism - Example

EG: Prove that 

is isomorphic to  .

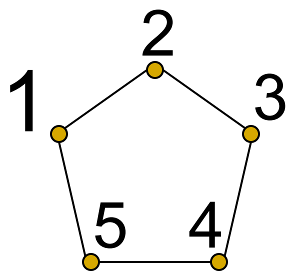
First label the vertices:

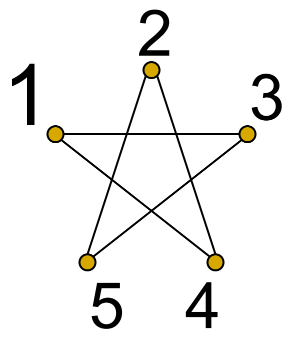




Graph Isomorphism - Example

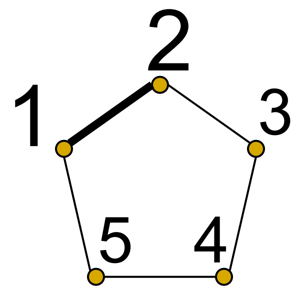
Next, set *f* (1) = 1 and try to walk around clockwise on the star.

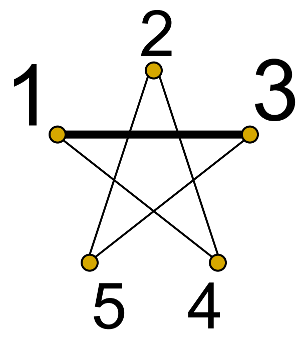




Graph Isomorphism - Example

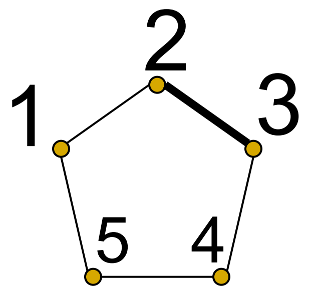
Next, set *f* (1) = 1 and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set *f* (2) = 3.

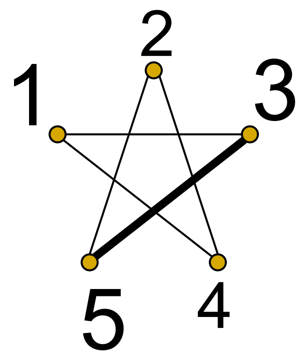




Graph Isomorphism - Example

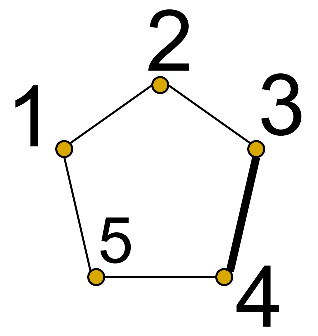
Next, set *f* (1) = 1 and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set *f* (2) = 3. Next vertex is 5 so set *f* (3) = 5.

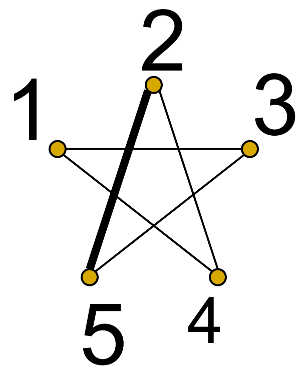




Graph Isomorphism - Example

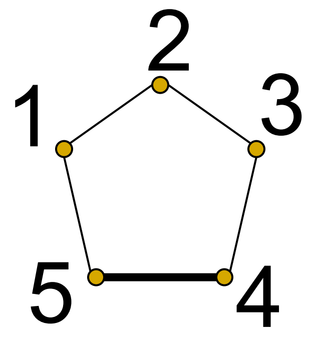
Next, set *f* (1) = 1 and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set *f* (2) = 3. Next vertex is 5 so set *f* (3) = 5. In this fashion we get *f* (4) = 2

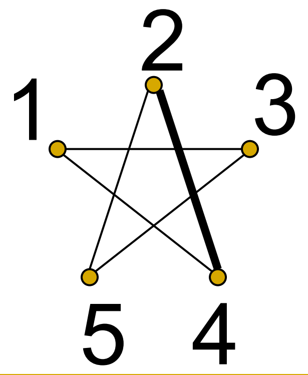




Graph Isomorphism - Example

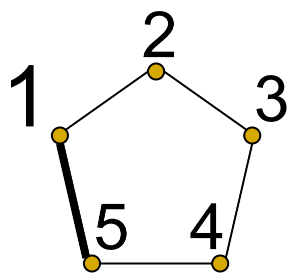
Next, set $f(1) = 1$ and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set $f(2) = 3$. Next vertex is 5 so set $f(3) = 5$. In this fashion we get $f(4) = 2$, $f(5) = 4$.

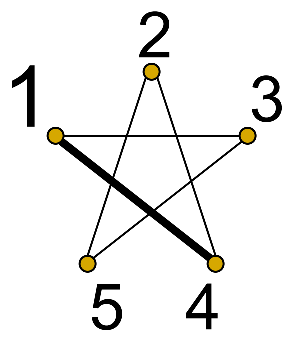




Graph Isomorphism - Example

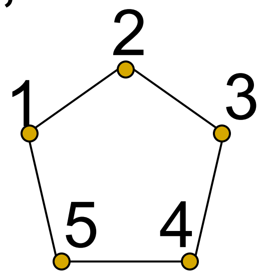
Next, set *f* (1) = 1 and try to walk around clockwise on the star. The next vertex seen is 3, *not* 2 so set *f* (2) = 3. Next vertex is 5 so set *f* (3) = 5. In this fashion we get *f* (4) = 2, *f* (5) = 4. If we would continue, we would get back to *f* (1) =1 so this process is well defined and *f*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

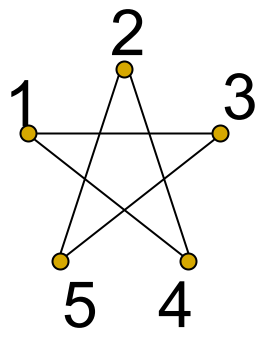




Graph Isomorphism - Example

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

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 -Negative Examples

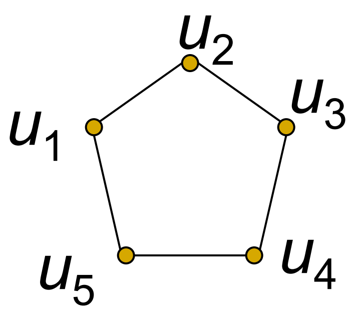
Once you see that graphs are isomorphic, it is easy to prove it.

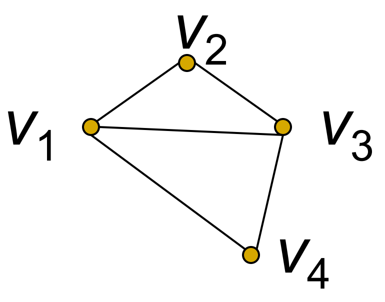
To prove that two graphs are not isomorphic is usually more difficult.

- Need to show that no function can exist that satisfies defining properties of isomorphism.
- In practice, you try to find some intrinsic property that differs between the 2 graphs in question.

Graph Isomorphism -Negative Examples

A: Why are the following non-isomorphic?

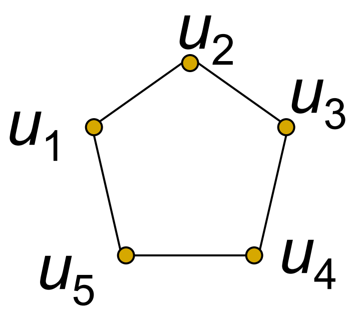


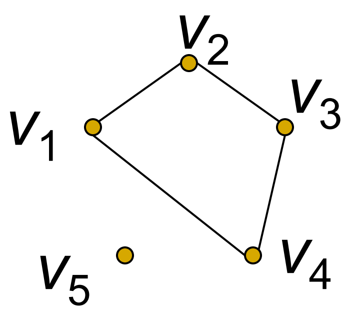


Graph Isomorphism -Negative Examples

A: 1st graph has more vertices than 2nd.

Q: Why are the following non-isomorphic?

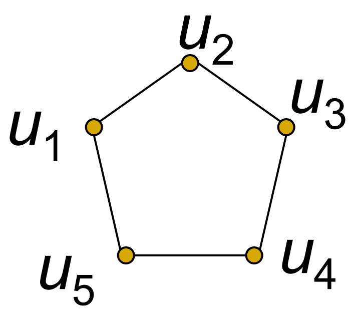


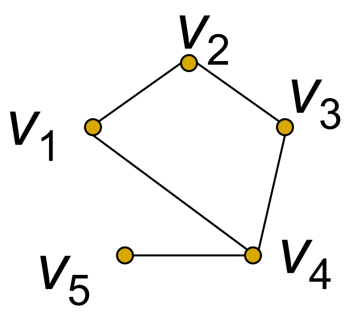


Graph Isomorphism -Negative Examples

A: 1st graph has more edges than 2nd.

Q: Why are the following non-isomorphic?

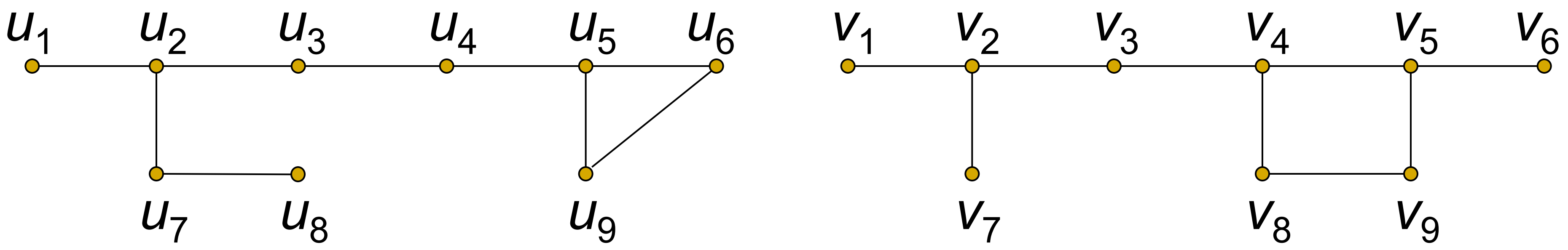




Graph Isomorphism -Negative Examples

A: 2nd graph has vertex of degree 1, 1st graph doesn't.

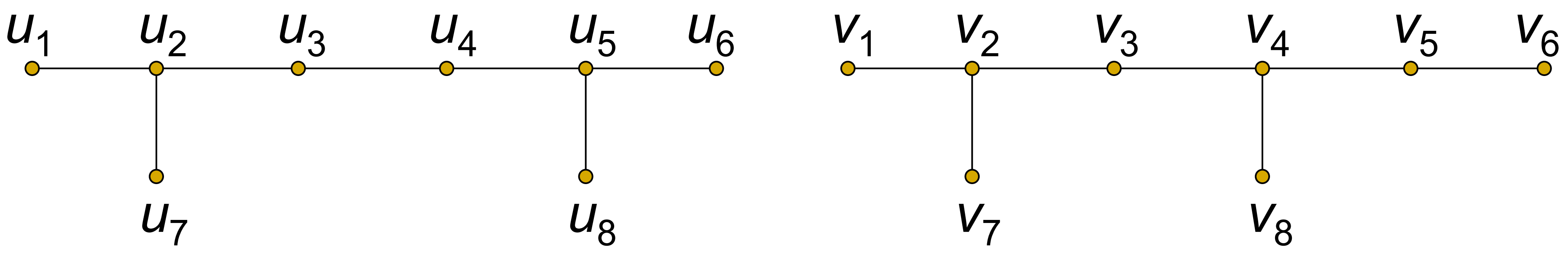
Q: Why are the following non-isomorphic?



Graph Isomorphism -Negative Examples

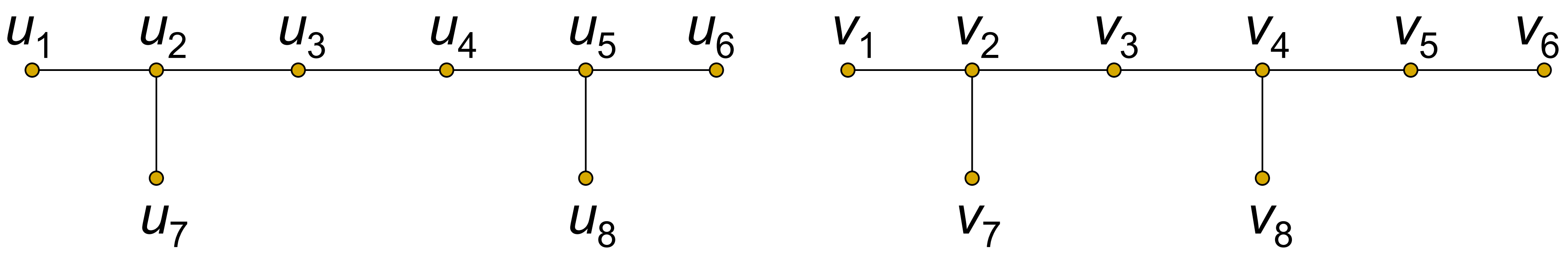
A: 1st graph has 2 degree 1 vertices, 4 degree 2 vertex and 2 degree 3 vertices. 2nd graph has 3 degree 1 vertices, 3 degree 2 vertex and 3 degree 3 vertices.

Q: Why are the following non-isomorphic?



Graph Isomorphism -Negative Examples

A: None of the previous approaches work as there are the same no. of vertices, edges, and same no. of vertices per degree.

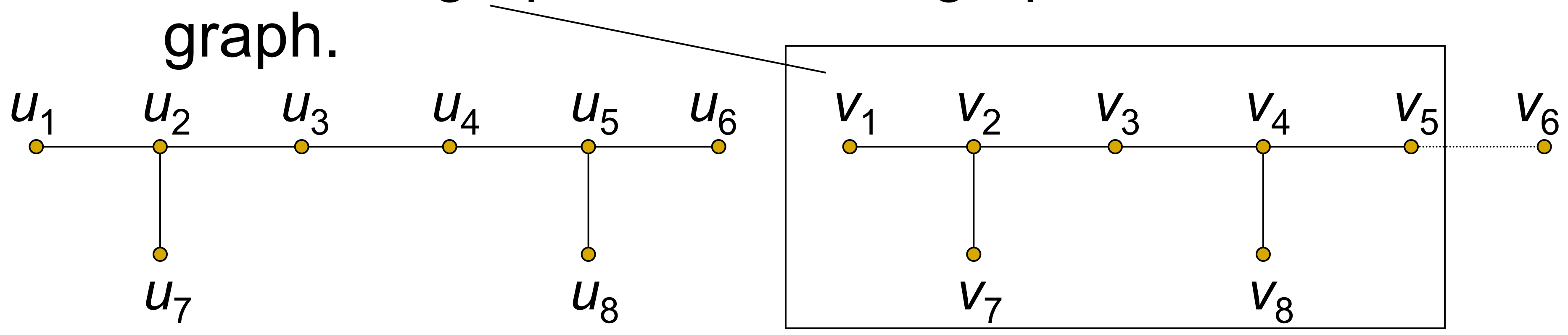


LEMMA: If *G* and *H* are isomorphic, then any subgraph of *G* will be isomorphic to some subgraph of *H*.

Q: Find a subgraph of 2nd graph which isn't a subgraph of 1st graph.

Graph Isomorphism -Negative Examples

A: This subgraph is not a subgraph of the left graph.



Why not? Deg. 3 vertices must map to deg. 3 vertices. Since subgraph and left graph are symmetric, can assume v_2 maps to u_2. Adjacent deg. 1 vertices to v_2 must map to degree 1 vertices, forcing the deg. 2 adjacent vertex v_3 to map to u_3. This forces the other vertex adjacent to v_3, namely v_4 to map to u_4. But then a deg. 3 vertex has mapped to a deg. 2 vertex→← ❷