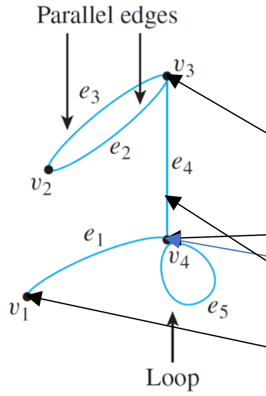
CS1231S Discrete Structures
Notes on Graphs and Trees



A **graph** G = {V, E} consists of

- V(G), a finite, nonempty set of **vertices**; and E(G), a finite set of **edges**;

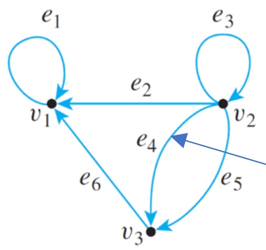
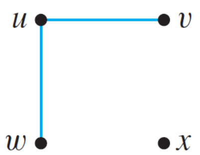
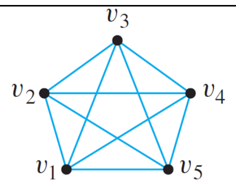
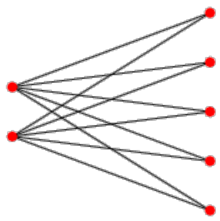
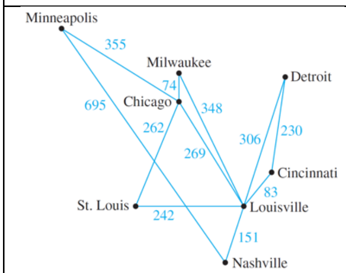
Each **edge** e = {v, w} is associated with a set consisting of one or two vertices called its **endpoints** in which the edge connects.

Adjacent vertices: v_3 and v_4 are connected by an edge e_4.

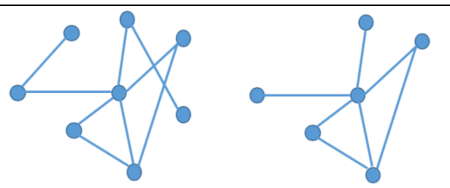
Adjacent to itself: v_4 is an endpoint to a loop.

Adjacent edge: e_1 and e_4 are incident on the same endpoint v_4.

Types of Graphs

	**Directed Graph** Instead of E(G), we have D(G), a set of **directed edges**; Each edge e = (v, w) is associated with an **ordered pair** of endpoints. **Directed edges**: e_4 is the directed edge from v_2 and v_3.
	**Simple Graph** An <u>undirected graph</u> that <u>does not have any loops or parallel edges</u>. Maximum number of edges in a simple graph with n vertices = $\binom{n}{2}$
 K_5	**Complete Graph** (on n vertices) K_n is a <u>simple graph</u> with n vertices and <u>exactly one edge connecting each pair</u> of distinct vertices. Every edge will have a degree of n − 1.
 $K_{2,5}$	**Complete Bipartite Graph** (on (m, n) vertices) $\boldsymbol{K_{m,n}}$ is a simple graph, • with distinct vertices $\boxed{v_1, v_2, \cdots, v_m}$ and $\boxed{w_1, w_2, \cdots, w_n}$ For all i, k = 1, 2, ..., m and for all j, l = 1, 2, ..., n, 1. There is an edge from each vertex v_i to each vertex w_j. 2. There is no edge from any vertex v_i to any other vertex v_k. 3. There is no edge from any vertex w_j to any other vertex w_l.
	**Weighted Graph** is a graph for which each edge has an associated positive real number weight. • **w(e)** denotes the weight of e, and **w(G)** denotes the total weight of the graph.

Subgraphs



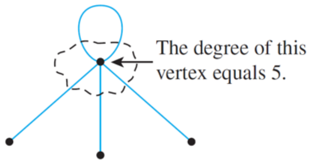
Graph G H, a subgraph of G

A graph H is said to be a **subgraph** of graph G if, and only if,

- every vertex in H is also a vertex in G;
- every edge in H is also an edge in G;
- every edge in H has the same endpoints as it has in G.

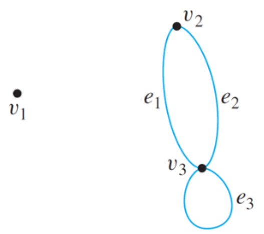
Degree

Degree of a vertex v, **deg(v)**



The number of edges that are incident on v, with an edge that is a loop counted twice.

Total degree of a graph G

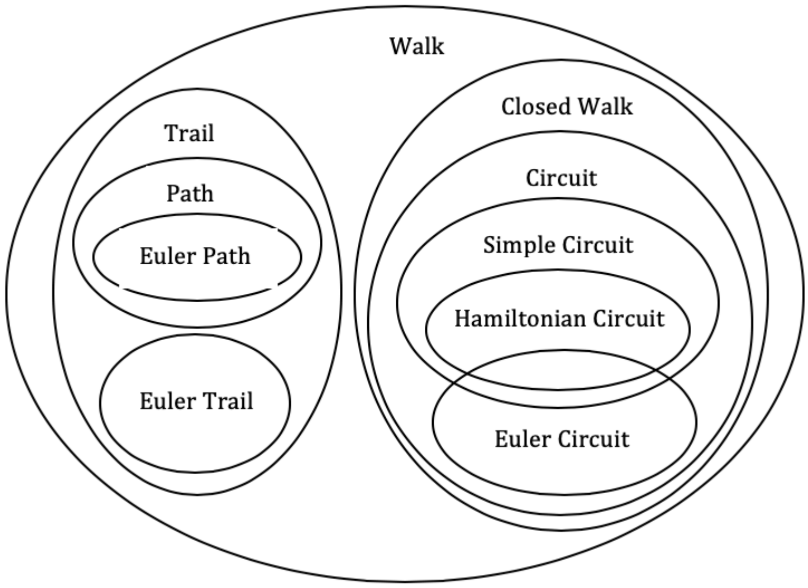


The sum of all the degrees of all the vertices in G.
Total degree of G $= deg(v_1) + deg(v_2) + deg(v_3)$

Handshake Theorem:

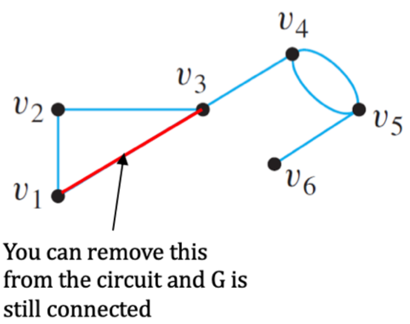
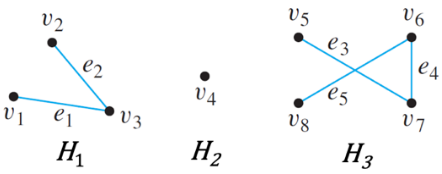
- Total degree of G $= 2 \times$ Number of edges in G.
- Total degree of a graph is always even.
- In any graph, there is an even number of vertices of odd degree.

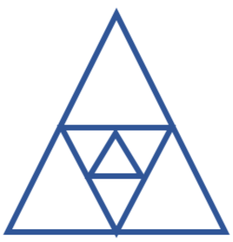
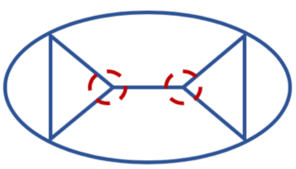
Travel

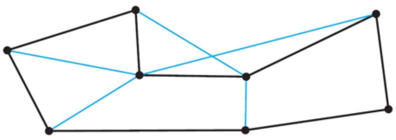


Type of Travel	Repeated Edge?	Repeated Vertex?	Starts and ends at the same point?	Must contain at least one edge?
Walk	Allowed	Allowed	Allowed	Not required
Trail	Not allowed	Allowed	Allowed	Not required
Path	Not allowed	Not allowed	Not allowed	Not required
Closed Walk	Allowed	Allowed	Required	Not required
Circuit	Not allowed	Allowed	Required	Required
Simple Circuit	Not allowed	Only 1st and last	Required	Required
Euler Circuit	Not allowed	Allowed	Required	Required
	+ Contains every vertex and every edge of G.			
Euler Trail / Path	Not allowed	Depends if it is a trail / path		Not required
	+ Passes through every vertex at least once			

	+ Passes through every edge exactly once			
Hamiltonian Circuit	Not allowed	Only 1^{st} and last	Required	Required
	+ Contains every vertex of G.			

Connectedness	
<u>Connected Vertices</u>	Two vertices v and w are connected iff there is a walk from v to w.
Connected Graphs 	The graph G is connected iff given any two vertices v and w in G, there is a walk from v to w. **Connected graph:** $\forall$ *vertices* $v, w \in V(G), \exists$ a walk from v to w. <u>What happens when a graph G is connected?</u> 1. **Paths everywhere!** Any two distinct vertices in G can be connected by a path. 2. **Edge removability!** If edges v and w are part of a circuit, and one edge is removed from the circuit, ○ There still exists a trail from v to w in G, ○ G will still be connected.
Connected Components 	A connected component of a graph is a connected subgraph of the largest possible size. A graph H is a **connected component** of a graph G if, and only if, 1. The graph H is a subgraph of G; 2. The graph H is connected; and 3. No connected subgraph of G has H as a subgraph and contains vertices or edges that are not in H.

Euler and Hamilton	
 An Eulerian Graph.	**Euler circuit** for G is a circuit that contains every vertex and every edge of G. • Has at least one edge, uses every edge of G exactly once; • Starts and ends at the same vertex, uses every vertex of G at least once. **Graph has an Euler Circuit** $\Leftrightarrow$ Graph is connected and every vertex has a positive even degree $\Rightarrow$ Every vertex has a positive even degree **Contrapositive:** Some vertex has an odd degree $\Rightarrow$ Graph does not have an Euler Circuit.
 There is an Euler Trail between the vertices of odd degree.	**Euler trail/path** from v to w *(distinct vertices)* starts at v, ends at w, ○ Passes through every vertex of G at least once. ○ Traverses every edge of G exactly once. Adding an edge between the two vertices v and w will give an Euler Circuit. **There is an Euler Trail from vertices v to w in graph G** $\Leftrightarrow$ Graph is connected, v and w have odd degree, and all other vertices have a positive even degree.



H is indicated by the black lines.

Hamiltonian circuit for G is a simple circuit that includes every vertex of G.

- Every vertex of G appears exactly once, except for the first and the last, which are the same.

Graph has an Hamiltonian Circuit

⇒ G has a subgraph H with the following properties:

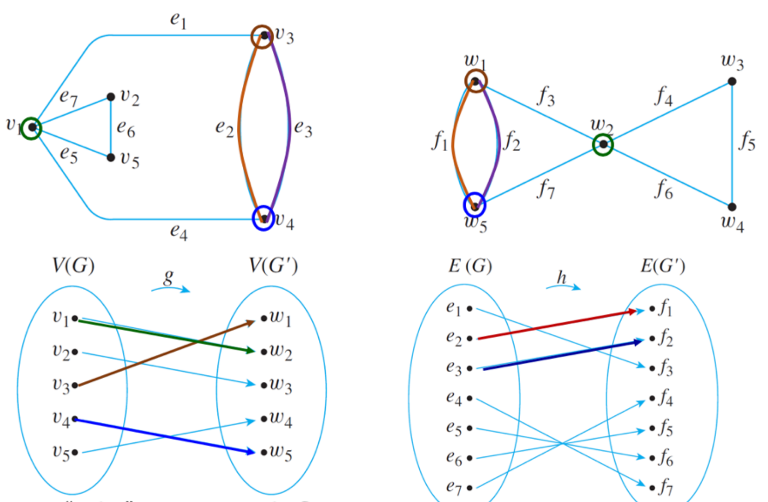
1. H contains every vertex of G.
2. H is connected.
3. H has the same number of edges as vertices.
4. Every vertex of H has degree 2.

Contrapositive: If a graph G does *not* have a subgraph H with properties (1)–(4), then G does *not* have a Hamiltonian circuit.

For this section, I have omitted most of the formulae regarding matrix manipulation because it is covered in MA1101R Linear Algebra.

Adjacency Matrix	
Of a directed graph	a_{ij} = Number of arrows from v_i to v_j
Of an undirected graph	a_{ij} = Number of edges connecting v_i to v_j The matrix is symmetric.
Of a graph with connected components $G_1, G_2, \cdots, G_k$	$$\begin{pmatrix} A_1 & O & \cdots & O \\ O & A_2 & \cdots & O \\ \vdots & \vdots & \ddots & \vdots \\ O & O & \cdots & A_k \end{pmatrix}$$ Each A_i is an $n_i \times n_i$ adjacency matrix for G_i, and the Os represent matrices whose entries are all 0s.
Counting Walks of Length n	
Number of walks of length n from v_i to v_j is the (i,j)-entry of $\mathbf{A}^{\mathbf{n}}$, where $\mathbf{A}$ is the adjacency matrix.	

Graph Isomorphism



g "pairs" every vertex in G to a distinct vertex in G'

h "pairs" every edge in G to a distinct edge in G'

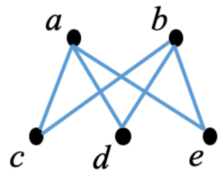
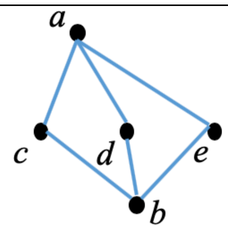
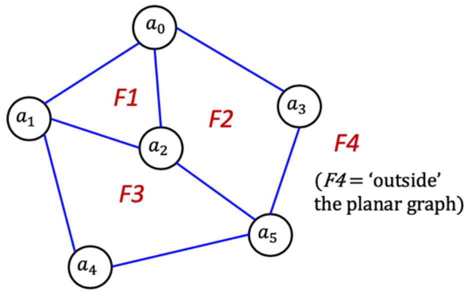
G is isomorphic to G' if, and only if,

- there exist one-to-one correspondences $g: V(G) \to V(G')$ and h: $E(G) \to E(G')$; and
- for all vertices and endpoints in G, v is an endpoint of e ⇔ g(v) is an endpoint of g(e).

Let S be a set of graphs and let R be the relation of graph isomorphism on S. Then R is an equivalence relation on S.

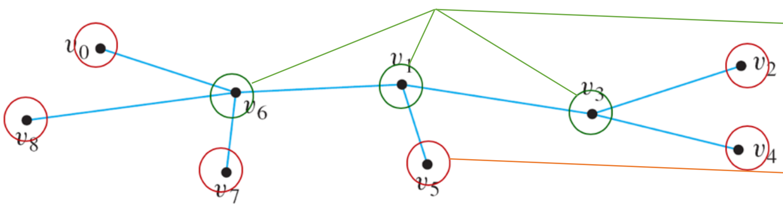
- Graph isomorphism is symmetric, reflexive and transitive.

Planar Graphs

 Non-planar representation  Planar representation	**Planar graph** A graph that can be drawn on a (two-dimensional) plane without edges crossing.
	

A graph G is a **tree**
⇔ it is circuit-free and connected
⇔ G is a connected graph with n vertices and $n-1$ edges.

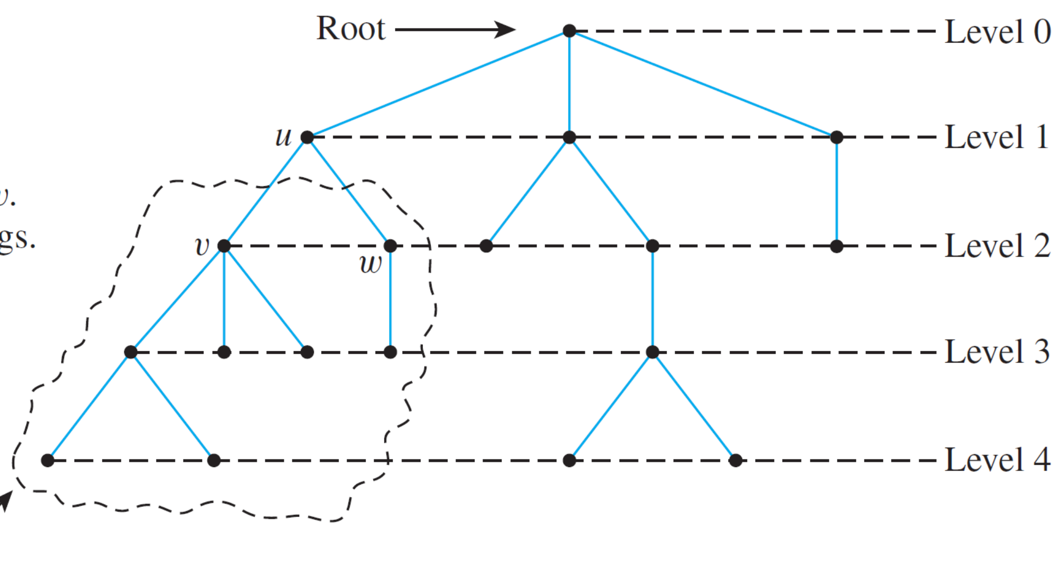
A graph is a **forest** if, and only if, it is circuit-free and not connected.



Internal vertices: Vertices of degree greater than 1.

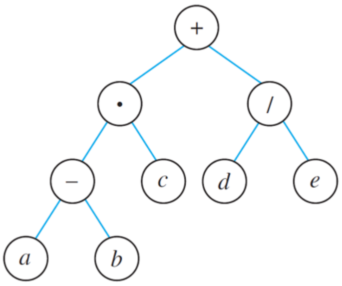
Terminal vertices: Vertices of degree 1.

Properties of Trees	
Vertices of a tree	Any non-trivial tree (more than one vertex) has at least one vertex of degree 1 (terminal vertex).
Edges of a tree	Any tree with n vertices has n − 1 edges.

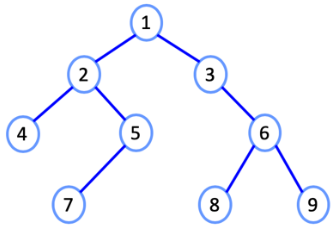
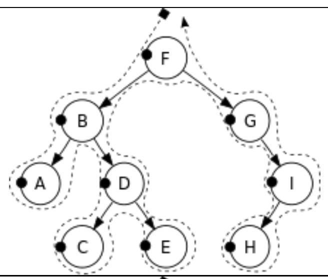
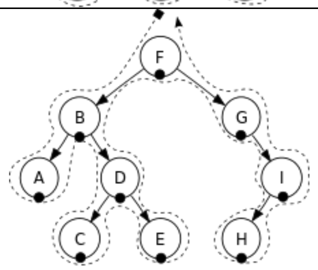


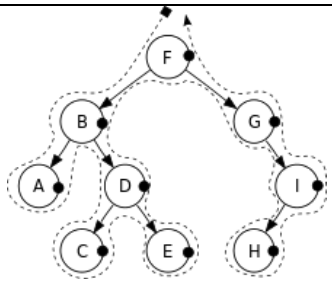
v is a child of u.
u is the parent of v.
v and w are siblings.

Vertices in the enclosed region are descendants of u, which is an ancestor of each.

Types of Trees	
Rooted Tree	There is 1 vertex, the root, that is distinguished from the others.
Binary Tree	Rooted tree where every parent has at most 2 children. **Relation between its height (h) and number of terminal vertices (t)** $t \leq 2^h \equiv log_2 t \leq h$
Full Binary Tree 	A binary tree in which each parent has exactly two children. **Full Binary Tree Theorem:** If T is a full binary tree with k internal vertices, then T has a total of $2k + 1$ vertices and $k + 1$ terminal vertices.
Spanning Tree (for a graph G)	Subgraph of G that contains every vertex of G and is a tree. • Any two spanning trees for a graph has the same number of edges.
Minimum Spanning Tree (for a connected weighted graph)	Spanning tree that has the least possible total weight compared to all other spanning trees for the graph.

Algorithms for Constructing a Minimum Spanning Tree	
Kruskal's Algorithm	Examine the edges of the graph one-by-one from the smallest weight to the largest weight. At each stage of examination, add the edge to the minimum spanning tree only if this addition **does not create a circuit.**
Prim's Algorithm	Pick any vertex in the graph. Then, choose the smallest edge that connects the vertex to another unvisited vertex.

Binary Tree Traversal	
	**Breadth-First Search** Starts at the root and visits its adjacent vertices, and then moves to the next level.
	**Pre-Order Depth-First Search** • Print the data of the root (or current vertex). • Traverse the left subtree by recursively calling the pre-order function. • Traverse the right subtree by recursively calling the pre-order function.
	**In-Order Depth-First Search** ▪ Traverse the left subtree by recursively calling the in-order function ▪ Print the data of the root (or current vertex) ▪ Traverse the right subtree by recursively calling the in-order function



Post-Order Depth-First Search

- Traverse the left subtree by recursively calling the post-order function
- Traverse the right subtree by recursively calling the post-order function
- Print the data of the root (or current vertex)

Table of Edgy Vertices

	Edges, $\|E\|$	**Vertices, $\|V\| = n$**
Graphs	Maximum $\|E\| = \binom{n}{2}$ Total degree of the graph $= 2 \times \|E\|$ and is always even.	There is an even number of vertices with an odd degree.
Complete Graph	Every edge has a degree of $n - 1$.	
Eulerian Graphs		Every vertex has a positive even degree.
Non-Eulerian Graphs		Some vertex has an odd degree.
Hamiltonian Graphs	Graph has a subgraph where $\|E\| = \|V\|$.	Every vertex of the subgraph H has degree of 2.
Planar Graphs	Number of faces $= Number\ of\ edges\ \|E\| - Number\ of\ vertices\ \|V\| + 2$	
Non-Trivial Trees	$\|E\| = n - 1$ A graph who fulfils the property above is a tree iff the graph is connected.	Has at least one vertex of degree 1
Full Binary Tree		k: Number of internal vertices t: Number of terminal vertices $\|V\| = 2k + 1$ $t = k + 1$ $t \leq 2^h \equiv log_2\, t \leq h$
Spanning Trees	Any two spanning trees for a graph has the same number of edges.	