Graphs

CS 330: Discrete Structures
A graph consists of a non-empty set of vertices (aka nodes) $V$, and a set of edges $E$ that describe connections between pairs of vertices. We typically draw graphs using "dots and lines" notation.

**Example:**

- $V = \{a, b, c, d, e\}$
- $E = \{\{a, b\}, \{b, c\}, \{b, d\}, \{c, e\}, \{d, e\}\}$
- $G = (V, E)$
In a directed graph, edges are ordered pairs (i.e., order matters). We draw such graphs using arrows to indicate the direction.

**Example:**

- **Vertices (V):** \{a, b, c, d, e\}
- **Edges (E):** \{(a, b), (b, e), (c, d), (d, c)\}
- **Graph (G):** \((V, E)\)

**Distinct Edges:** Sometimes, written \(<c\rightarrow d>, <d\rightarrow c>\)
- an edge that connects a vertex to itself is a loop e.g. 

- a graph that permits multiple edges between the same nodes is a multigraph.

- a graph that contains no loops and is not a multigraph is a simple graph

--- in this class when we use the term "graph", we will mean "undirected simple graph" (there is no real consensus on graph terminology/notation in the wild)
- two vertices in a graph are adjacent if they are connected by an edge.
- an edge is incident with the vertices that it connects.
- the degree of a vertex is the number of edges incident with it.

![Diagram](image_url)

**Example:**

- adjacent vertices: $a+ b, b+ c, b+ d, c+ d$

  - $\text{deg}(a) = 1$
  - $\text{deg}(b) = 3$
  - $\text{deg}(c) = \text{deg}(d) = 2$
“Handshaking” theorem: given a graph $G = (V,E)$,

$$\sum_{w \in V} \deg(w) = 2|E|$$

E.g. $|E| = \frac{10}{2} = 5$

sum of degrees $= 1 + 3 + 3 + 2 + 1 = 10$
a subgraph of $G = (V, E)$ is a graph $H = (W, F)$ where $W \subseteq V$ and $F \subseteq E$.

A subgraph of $G = (V, E)$ induced by $V' \subseteq V$ is the graph $G' = (V', E')$, where $E'$ contains only edges from $E$ that connect pairs of vertices in $V'$.
e.g., draw three distinct subgraphs of \( G = (V,E) \), where

\[
V = \{ a, b, c, d \}, \quad E = \{ \langle a-b \rangle, \langle a-c \rangle, \langle b-c \rangle, \langle d-a \rangle \}
\]

\[ \begin{align*}
\begin{array}{c}
\text{a} \\
\downarrow \\
\text{d}
\end{array} & \begin{array}{c}
\text{b} \\
\downarrow \\
\text{c}
\end{array} & \begin{array}{c}
\text{a} \\
\downarrow \\
\text{c}
\end{array} & \begin{array}{c}
\text{d} \\
\downarrow \\
\text{c}
\end{array}
\end{align*} \]
SPECIAL GRAPHS/
SUBGRAPHS

Complete graph:
- Has an edge between every pair of vertices.
- $n\geq 2$ vertices $v_1, v_2, v_3, \ldots, v_n$.
- Edges $\{<v_1, v_2>, <v_2, v_3>, \ldots, <v_{n-1}, v_n>\}$.

Empty graph:
- No edges.

Line/path:
- $n\geq 2$ vertices $v_1, v_2, v_3, \ldots, v_n$.
- Edges $\{<v_1, v_2>, <v_2, v_3>, \ldots, <v_{n-1}, v_n>\}$.

Circuit:
- $n\geq 3$ vertices $v_1, v_2, v_3, \ldots, v_n$.
- Edges $\{<v_1, v_2>, <v_2, v_3>, \ldots, <v_n, v_1>\}$.

Empty graph:
- No edges.

Bipartite graph:
- If its vertices can be partitioned into sets $V_1 \cap V_2 = \emptyset$, every edge connects a vertex from $V_1$ to a vertex from $V_2$.
what do we model using graphs?

- computer networks
- transportation networks
- molecular structures
- social networks
- circuit layouts
- resource/scheduling interdependencies
- and much, much more!
e.g. can any two nodes in this network communicate?

- interested in the connectedness of the graph
- is there a path between any two nodes? (not in this example!)
e.g. how robust are these networks to failure?

— i.e., how many nodes/edges do we have to remove to disconnect the remaining nodes?

looking for out vertices/edges of the graph.
e.g., are the pairs of graphs below the same (after some stretching/compressing/ twisting, but without adding/removing vertices/edges)?

- \[
\begin{array}{llll}
\text{a} & \text{b} & \text{c} & \text{d} \\
\text{e} & \text{f} & \text{g} & \text{h}
\end{array}
\]

- \[
\begin{array}{llll}
\text{a} & \text{b} & \text{c} & \text{d} \\
\text{e} & \text{f} & \text{g} & \text{h}
\end{array}
\]

Question of graph isomorphism:

Applications in chemistry — are two molecules of the same formula structurally identical?

Chip fabrication/optimization — is a particular layout of components equivalent to the original schematic?
e.g., in the U.S., do men or women, on average, have more opposite sex partners? (only considering heterosexual relationships)

- Laumann et al. @ UChicago, 1994: men have 74% more opposite sex partners.

Men | Women
--- | ---
\[ \sum_{v \in \text{Men}} \deg(v) = \sum_{v \in \text{Women}} \deg(v) = |E| \]

Avg # partner for men = \[ \frac{|E|}{|\text{Men}|} \]

Avg # partner for women = \[ \frac{|E|}{|\text{Women}|} \]

ratio = \[ \frac{|\text{Women}|}{|\text{Men}|} \]

In U.S., \[ \frac{|\text{Women}|}{|\text{Men}|} = \frac{161 \text{ million}}{156.1 \text{ million}} \approx 1.031 \]

(men more promiscuous by 3.1%)

(another bipartite graph)
e.g., can the following circuit schematic be redrawn in such a way so that none of the wires (edges) overlap?

- goal: **planar representation** of a graph

- useful in many other problems; e.g., layout of transportation networks (road/rail)
e.g., how many separate timeslots do we need to avoid any final exam scheduling conflicts, given that the graph below contains edges between all classes of overlapping student enrollments?

3 timeslots are sufficient

"graph coloring" problem

famous result: need at most 4 colors to color a planar graph.

this clique makes it impossible to do less than 3
Other common graph representations

- **Adjacency list**: for each vertex, list all adjacent vertices.

  - **Example**:
    - **a**: b, c, e
    - **b**: a, c, d
    - **c**: a, b
    - **d**: b
    - **e**: a

- **Adjacency matrix**: $m_{ij} = 1$ if edge $(v_i, v_j)$ exists, 0 otherwise.

  - **Example**:
    - **a**
      - b: 1, c: 1, e: 1
    - **b**
      - a: 1, c: 1, d: 1
    - **c**
      - a: 1, b: 1
    - **d**
      - b: 1
    - **e**
      - a: 1

- **Incidence matrix**: $m_{ij} = 1$ if edge $j$ is incident to vertex $i$, 0 otherwise.

  - **Example**:
    - **a**
      - e: 1, e_2: 1, e_3: 1, e_4: 1, e_5: 1
    - **b**
      - e: 1, e_2: 1, e_3: 1, e_4: 1, e_5: 1
    - **c**
      - e: 1, e_2: 1, e_3: 1
    - **d**
      - e: 1, e_2: 1, e_3: 1
    - **e**
      - e: 1

- **Given graph**:

  - Vertices: a, b, c, d, e
  - Edges: e_1, e_2, e_3, e_4, e_5

  - **Adjacency list**:
    - **a**: e_1, e_3, e_5
    - **b**: e_1, e_2, e_3
    - **c**: e_2, e_3
    - **d**: e_5
    - **e**: e_1, e_2, e_3, e_4, e_5

- **Adjacency matrix**:

  - **a**
    - b: 1, c: 1, e: 1
  - **b**
    - a: 1, c: 1, d: 1
  - **c**
    - a: 1, b: 1
  - **d**
    - b: 1
  - **e**
    - a: 1

- **Incidence matrix**:

  - **a**
    - e_1: 1, e_2: 1, e_3: 1, e_4: 1, e_5: 1
  - **b**
    - e_1: 1, e_2: 1, e_3: 1
  - **c**
    - e_2: 1, e_3: 1
  - **d**
    - e_5: 1
  - **e**
    - e_1: 1, e_2: 1, e_3: 1, e_4: 1, e_5: 1
Isomorphism

Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijection $f: V_1 \to V_2$, such that:

$\forall u, w \in V_1$, edge $\langle u-w \rangle \in E_1$ iff $\langle f(u)-f(w) \rangle \in E_2$
e.g., are \( G_1 = \{a, b, c, d\} \) and \( G_2 = \{x, y, z\} \) isomorphic?

\[ G_1 = (V_1 = \{a, b, c, d\}, E_1 = \{a-b, b-c, c-d, d-a\}) \]
\[ G_2 = (V_2 = \{w, x, y, z\}, E_2 = \{w-y, y-x, x-z, z-w\}) \]

function \( f : V_1 \to V_2 \), where:

\[
\begin{align*}
  f(a) &= w \\
  f(b) &= y \\
  f(c) &= x \\
  f(d) &= z
\end{align*}
\]

This gives us the needed bijection.

Yes — isomorphic.
e.g., are $G_1 = \begin{array}{c}
\text{a} \\
\text{d} \\
\text{c} \\
\text{b}
\end{array}$ and $G_2 = \begin{array}{c}
\text{w} \\
\text{z} \\
\text{x} \\
\text{y}
\end{array}$ isomorphic?

No! Bijection between vertices must also give us bijection between edges. $|E_1| \neq |E_2|$, so no bijection is possible.
the best algorithms for determining if two graphs are isomorphic have exponential worst case runtime complexity—however, the graph isomorphism problem is not in NP-complete.

* As of 2017, László Babai @ UChicago claims to have shown graph isomorphism can be solved in quasi-polynomial time — i.e., $2^{O((\log n)^c)}$.

** open source program "NAUTY" can test most graphs up vertices <100 for isomorphism in <1 second.
All graph properties we care about are preserved under isomorphism!

\[ \text{e.g.} \quad a \quad b \quad c \quad d \quad e \quad f \quad g \quad h \quad i \quad j \]

are structurally/semantically equivalent where all graph properties and algorithms are concerned.
Connectivity

- A graph is **connected** if there is a path between every pair of vertices.
- A **connected component** of a graph is a subgraph consisting of some vertex and all vertices and edges connected to it.

$$
\therefore \text{ a graph is connected iff it has a single connected component.}
$$

- A **cut vertex** (or edge, also known as articulation point or bridge), when removed from a graph, leaves more connected components than before. ('It disconnects the graph.')
e.g. for each graph, determine the number of connected components and identify cut vertices/edges, if they exist.
A graph \( G \) is \( k \)-vertex-connected if it has at least \( k \) vertices and removing any fewer than \( k \) vertices does not disconnect it.

- The vertex connectivity \( \kappa(G) \) is the max \( k \) s.t. \( G \) is \( k \)-vertex-connected, denoted \( \kappa \).

A graph \( G \) is \( k \)-edge-connected if removing any fewer than \( k \) edges does not disconnect it.

- The edge connectivity \( \lambda(G) \) is the max \( k \) s.t. \( G \) is \( k \)-edge-connected, denoted \( \lambda \).

For complete graphs of size \( n \), define:
- \( \kappa(K_n) = 0 \),
- \( \lambda(K_n) = n - 1 \).
e.g. determine $k(G)$ and $\lambda(G)$ for each graph below.

1. $k(G) = 2$
   $\lambda(G) = 2$

2. $k(G) = 3$
   $\lambda(G) = 3$

3. $k(G) = 1$
   $\lambda(G) = 2$

4. $k(G) = 1$
   $\lambda(G) = 1$

5. $k(G) = 2$
   $\lambda(G) = 2$

6. $k(G) = 2$
   $\lambda(G) = 3$

7. $k(G) = 3$
   $\lambda(G) = 3$
Graph Traversals

- a **path** in a graph of length $k$ consists of a sequence of vertices and edges $[v_0, e_1, v_1, e_2, v_2, \ldots, v_k]$ where $e_i = \langle v_{i-1}, v_i \rangle$

- a **circum** is a path that starts and ends on the same vertex

- a "**simple**" path/circuit does not contain duplicate edges

  (a.k.a. elsewhere "**path**" = no repeated edges/vertices, and "**cycle**" = circuit w/ no repeated edges/vertices)
"Bridge of Königsberg" (1736)

Euler: "can we start at some location, traverse all bridges once, and end up where we started?"
Euler: “can we start at some location, traverse all bridges once, and end up where we started?”
for graph $G = (V, E)$:

- an **Euler path** is a path that traverses every $e \in E$ once
- an **Euler circuit** is a path that traverses every $e \in E$ once

- what conditions are necessary/sufficient for a connected graph to have an Euler path/circuit?
e.g., construct an Euler circuit.

degrees of all vertices are even — necessary condition!
also sufficient
— for every node other than staff, we must leave after entering
— for staff node, we must re-enter + not leave at some point (but can revisit as needed)

(what about Euler paths?)
for graph $G = (V, E)$:

- a Hamiltonian path is a path that visits every $v \in V$ once.
- a Hamiltonian circuit is a circuit that visits every $v \in V$ once.

This makes it much harder!
Sufficient conditions:

Given \( G = (V, E) \),

**Dirac's theorem:** if \(|V| \geq 3\) and \( \forall w \in V \ \text{deg}(w) \geq \frac{|V|}{2} \),
then \( G \) has a Hamiltonian circuit.

**Ore's theorem:** if \(|V| \geq 3\) and \( \forall u, w \in V (\langle u-w \rangle \notin E) \)
\( \rightarrow \text{deg}(u)+\text{deg}(w) \geq |V| \),
then \( G \) has a Hamiltonian circuit.

---

---

---

---
the **Traveling Salesperson Problem (TSP)** is the problem of finding a Hamiltonian circuit in a *weighted* graph — i.e., one where a numerical "cost" (weight) is assigned to each edge — where the sum of edge weights in the circuit is minimized.
Both the Hamiltonian Circuit Problem (HCP) and TSP are NP-hard problems.

- how many Hamiltonian circuits might need to be considered to solve the TSP in a complete graph with $n$ vertices?

\[
\frac{(n-1)!}{2} \text{ distinct circuits} \quad \text{i.e., brute force approach } \in O(n!)
\]

- dynamic programming can find exact answer in $O(n^2 2^n)$

- approximation algorithms can find a solution for millions of nodes in short time with 3% of precise answer (most of time)
instead of a circuit, we often just care about the **shortest path** from a given node to some other node in a weighted graph.

```
e.g., shortest path from Chicago → Miami
    → LA
```
Dijkstra's algorithm is an algorithm for finding the shortest path distances in graph \( G = (V, E) \) with weights \( w(u, v) > 0 \) for all edges \( <u-v> \in E \), from some start node \( t \):

\[
\begin{align*}
S & \leftarrow \emptyset \\
\text{for all } u \in V: & \\
L(u) & \leftarrow \infty \\
L(t) & \leftarrow 0 \\
\text{while } S \neq V: & \\
\text{find node } u \notin S \text{ with minimum } L(u) & \\
S & \leftarrow S \cup \{u\} \\
\text{for all nodes } v \notin S \text{ adjacent to } u: & \\
\text{relax } (u, v) & \\
\end{align*}
\]

\[
\text{relax } (u, v): \\
\begin{cases}
\text{if } L(u) + w(u, v) < L(v): \\
L(v) = L(u) + w(u, v)
\end{cases}
\]
e.g. use Dijkstra's algorithm to find the shortest paths starting from node A in the following graph:

\[ S = \{ A, B, C, D \} \]
e.g. use Dijkstra's algorithm to find the shortest paths starting from node A in the following graph:

\[ S = \{A, B, D, C, E\} \]
Runtime complexity?

\[ S \leftarrow \emptyset \]

for all \( u \in V \):
\[ L(u) \leftarrow \infty \]

\[ L(t) \leftarrow 0 \]

while \( S \neq V \):

find node \( u \notin S \) with minimum \( L(u) \)
\[ S \leftarrow S \cup \{ u \} \]

for all nodes \( v \notin S \) adjacent to \( u \):
\[ \text{relax}(u, v) \]
Graph Coloring

- a coloring of a graph is an assignment of a color to each vertex s.t. no adjacent vertex has the same color.
- a graph that allows a coloring w/ k colors is "k-colorable"

- the chromatic number \( \chi(G) \) of graph \( G \) is the least # of colors needed to color the graph.

\* for any planar graph \( G \), \( \chi(G) \leq 4 \)

took over 100 years to prove! And only w/ assistance of computer-assisted checking of ~2000 cases. Still no computer-unassisted proof.
"greedy" basic coloring algorithm for graph $G = (V, E)$:

1. choose some ordering of vertices in $V$:
   \[ V_1, V_2, \ldots, V_n \]

2. order the colors: $C_1, C_2, \ldots$

3. for $i = 1, 2, \ldots, n$, assign $V_i$ the lowest possible color
   (i.e., s.t. no adjacent vertices have the same color)
e.g., apply the basic coloring algorithm to this map:

```
e.g. A B C D E

E D C B A
```

```
Note: we can draw a planar graph representation of any map
```
e.g., how many colors, at most, does the basic coloring algorithm assign to each of the following graphs?

- This basic algorithm seems to do pretty well, regardless of ordering!
e.g., give an example of a graph $G$ for which the basic coloring algorithm may assign substantially more colors than $\chi(G)$.

\[ \frac{|V|}{\chi(G)} \text{ colors assigned, only 2 needed!} \]
Conjecture: given graph $G = (V, E)$ where $\forall w \in V, \deg(w) \leq k$, the basic coloring algorithm assigns at most $k+1$ colors to $G$.

$|V| = n$ and

$P(M)$: (show $P(1) \land P(n) \Rightarrow P(n+1)$) — what is predicate $P$?

Basis: $P(1) =$ graph with $|V| = 1$: $\forall w \in V, \deg(w) = 0$, need 0+1 colors $\checkmark$

Inductive step: assume $P(n)$ is true (i.h.)

For $P(n+1)$, let $G = (V, E)$ with $|V| = n+1$, and $\forall w \in V, \deg(w) \leq k$

order nodes in $V$ as $w_1, w_2, \ldots, w_n, w_{n+1}$

by i.h., we know that this subgraph requires at most $k+1$ colors.
i.e. 1

\begin{itemize}
    \item n nodes, k+1 colors
    \item Max degree k
    \item because this node has at most k neighbors (up k distinct colors), \exists \text{color} \in \{c_1, c_2, \ldots, c_{k+1}\} such that we can assign to \text{w}_{n+1}
    \item \therefore P(n) \rightarrow P(n+1) \checkmark
\end{itemize}

QED.
in general, finding the chromatic number of a graph is NP-hard* — best algorithms have exponential runtime complexity.

deciding whether a graph is $k$-colorable for $k \geq 2$ is NP-complete (except for some special cases — e.g., 2-coloring — $O(|V|+|E|)$, $k \geq 3$ for planar graphs)