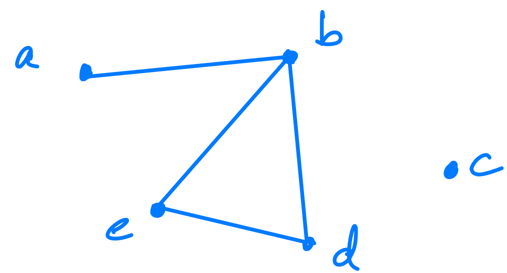
Graphs

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

e.g.



note: same as $\{d,b\}$ (order doesn't matter).

— sometimes written $\langle b\text{-}d \rangle$

$V = \{a, b, c, d, e\}$

$E = \{\{a,b\}, \{b,e\}, \{b,d\}, \{e,d\}\}$

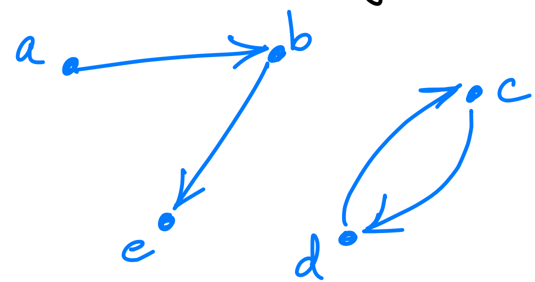
$G = (V, E)$

"digraph" — vs. undirected graph

in a **directed graph**, edges are ordered pairs (i.e., order matters.)

— we draw such graphs using arrows to indicate the direction.

e.g.,



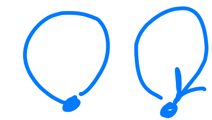
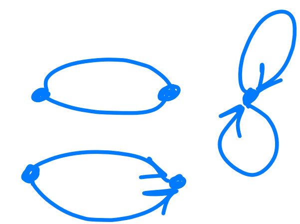
distinct edges

$V = \{a, b, c, d, e\}$

$E = \{(a,b), (b,e), (c,d), (d,c)\}$

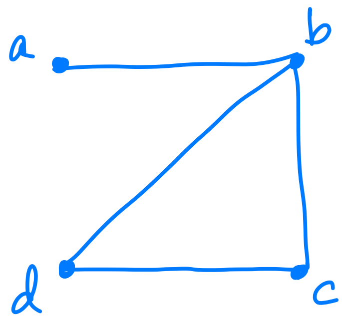
$G = (V, E)$

— sometimes written $\langle c \to d \rangle$, $\langle d \to c \rangle$

- 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**. e.g. 
- 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.

e.g.



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

$\deg(a) = 1$

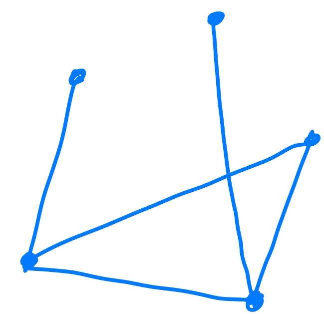
$\deg(b) = 3$

$\deg(c) = \deg(d) = 2$

"Handshaking" theorem: given a graph $G = (V, E)$,

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

e.g.



sum of degrees $= 1+3+3+2+1 = 10$

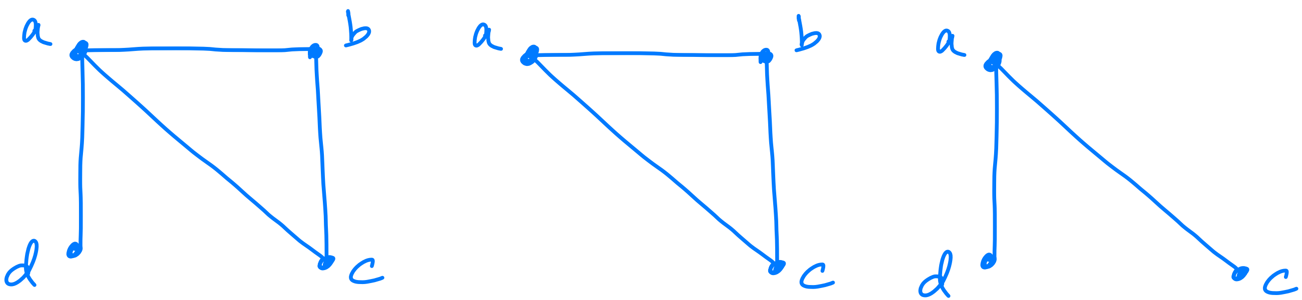
$|E| = 10/2 = 5$

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

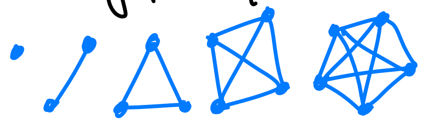
$V = \{a, b, c, d\}$, $E = \{\langle a-b \rangle, \langle a-c \rangle, \langle b-c \rangle, \langle d-a \rangle\}$



SPECIAL GRAPHS / SUBGRAPHS

complete graph

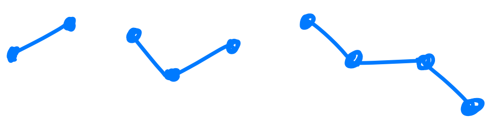
has an edge between every pair of vertices



(aka "clique" as a subgraph)

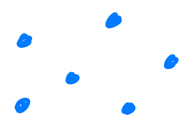
line / path

$n \geq 2$ vertices $v_1, v_2, v_3, \ldots, v_n$,
edges $\{\langle v_1 - v_2\rangle, \langle v_2 - v_3\rangle, \ldots, \langle v_{n-1} - v_n\rangle\}$



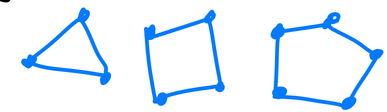
empty graph

no edges



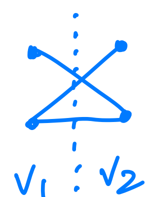
circuit

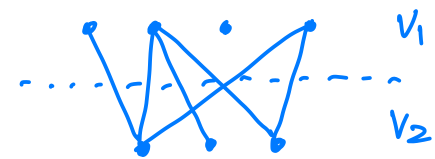
$n \geq 3$ vertices $v_1, v_2, v_3, \ldots, v_n$,
edges $\{\langle v_1 - v_2\rangle, \langle v_2 - v_3\rangle, \ldots, \langle v_n - v_1\rangle\}$



bipartite graph

if its vertices can be partitioned into sets V_1 & V_2 s.t. every edge connects a vertex from V_1 to a vertex from V_2

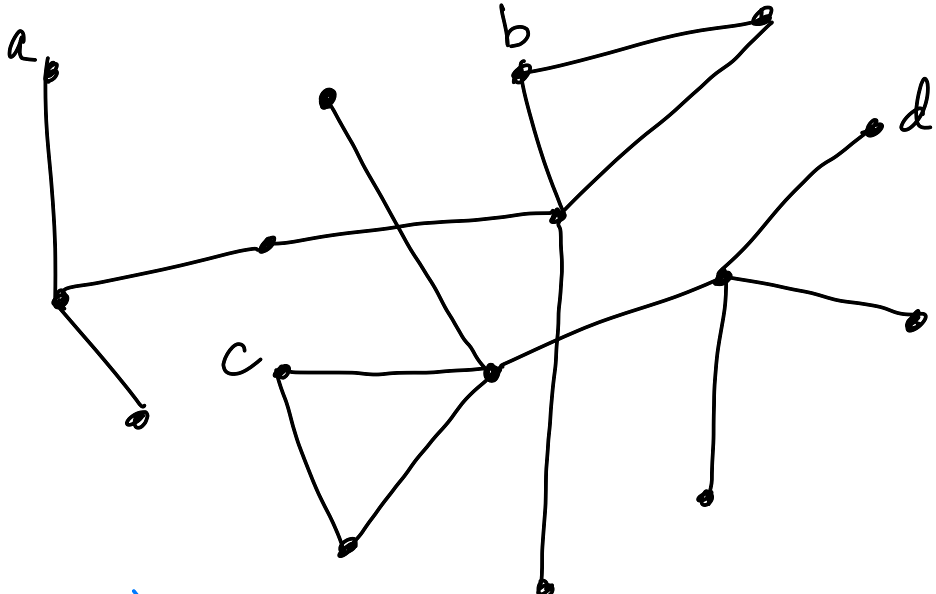




What do we model using graphs?

- computer networks
- transportation networks
- molecular structure
- social networks
- circuit layouts
- resource/scheduling interdependencies
- and much, much more!

e.g. can any two nodes in this network communicate?



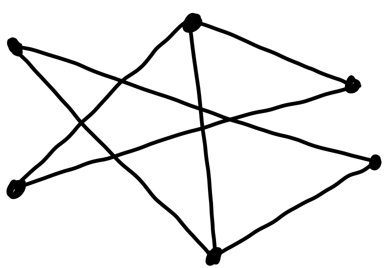
interested in the connectedness of the graph

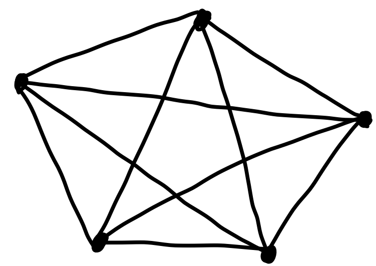
– is there a path between any two nodes?

(not in this example!)

e.g. how robust are these networks to failure?

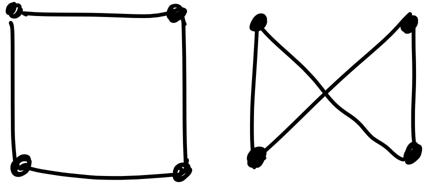
— i.e., how many nodes/edges do we have to remove to disconnect the remaining nodes?

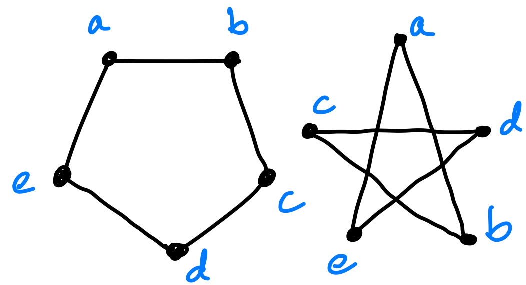


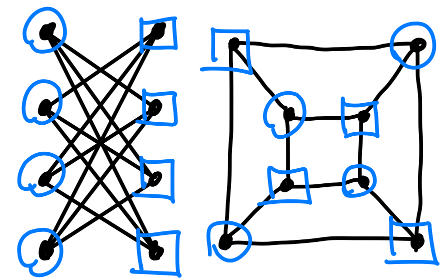


looking for <u>cut vertices/edges</u> of the graph.

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







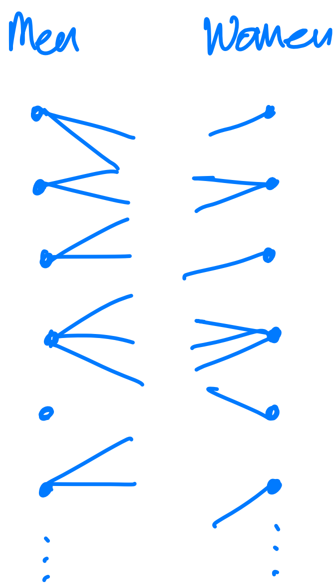
question of graph <u>isomorphism</u>

applications in chemistry — are two molecules w/ 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



(another bipartite graph)

because only considering opposite-sex partners:

$$\sum_{v \in \text{Men}} \deg(v) = \sum_{v \in \text{Women}} \deg(v) = |E|$$

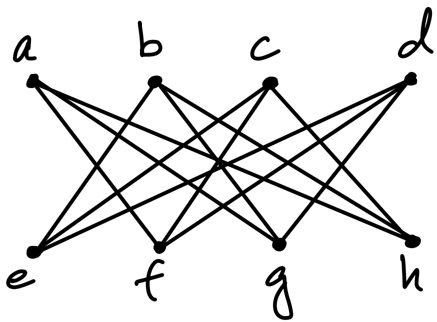
$$\text{avg \# partners for men} = \frac{|E|}{|\text{Men}|}$$

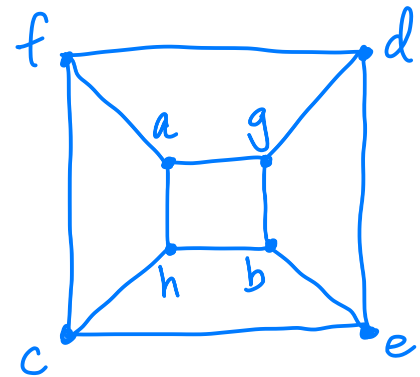
$$\text{'' for women} = \frac{|E|}{|\text{Women}|}$$

$$\text{ratio} = \frac{|\text{Women}|}{|\text{Men}|}$$

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%)

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

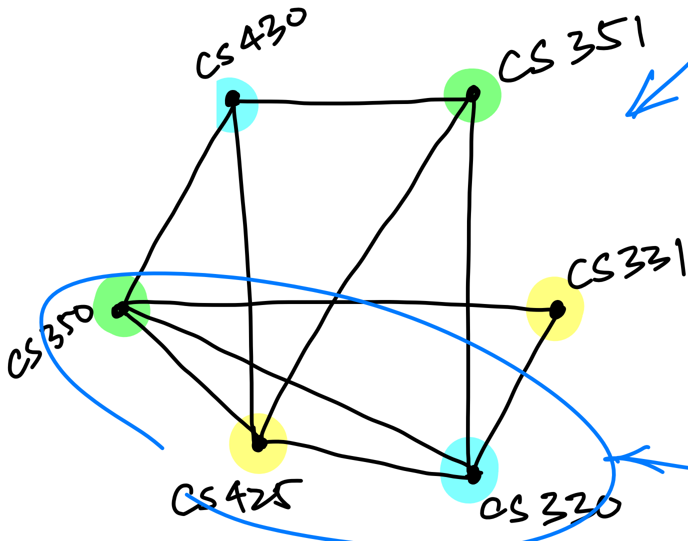




— goal: planar representation of a graph

— 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 w/ overlapping student enrollments?

3 timeslots are sufficient



"graph coloring" problem

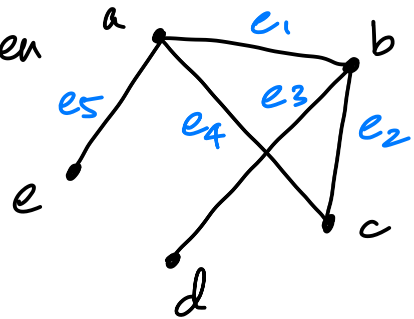
famous result:

need at most 4 colors to color a planar graph.

this clique makes it impossible to do less than 3

Other common graph representations

e.g., given



1. Adjacency lists: for each vertex, list all adjacent vertices

a	b, c, e
b	a, c, d
c	a, b
d	b
e	a

good when matrix reps are sparse

2. Adjacency matrix: $m_{ij} = 1$ if edge $\langle v_i - v_j \rangle$ exists, 0 otherwise

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

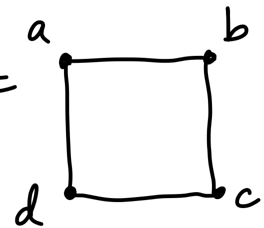
3. Incidence matrix: $m_{ij} = 1$ if edge j is incident w/ vertex i, 0 otherwise

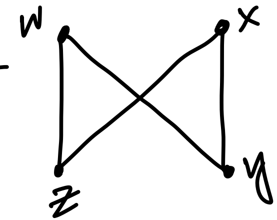
$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

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 $G_1 =$ 

and $G_2 =$ 

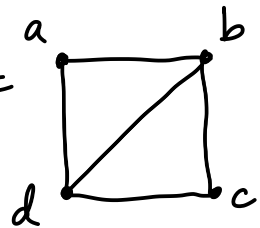
isomorphic?

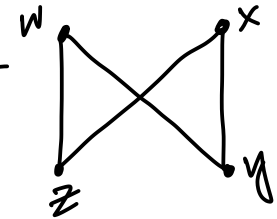
$$G_1 = \left(V_1 = \{a, b, c, d\},\ E_1 = \{\langle a\text{-}b\rangle, \langle b\text{-}c\rangle, \langle c\text{-}d\rangle, \langle d\text{-}a\rangle\}\right)$$

$$G_2 = \left(V_2 = \{w, x, y, z\},\ E_2 = \{\langle w\text{-}y\rangle, \langle y\text{-}x\rangle, \langle x\text{-}z\rangle, \langle z\text{-}w\rangle\}\right)$$

function $f: V_1 \to V_2$, where: $\begin{cases} f(a) = w \\ f(b) = y \\ f(c) = x \\ f(d) = z \end{cases}$ gives us the needed bijection.

Yes — isomorphic.

e.g., are $G_1 =$ 

and $G_2 =$ 

isomorphic?

No! Bijection between vertices must also give us bijection between edges. $|E_1| \neq |E_2|$, so no bijection is possible.

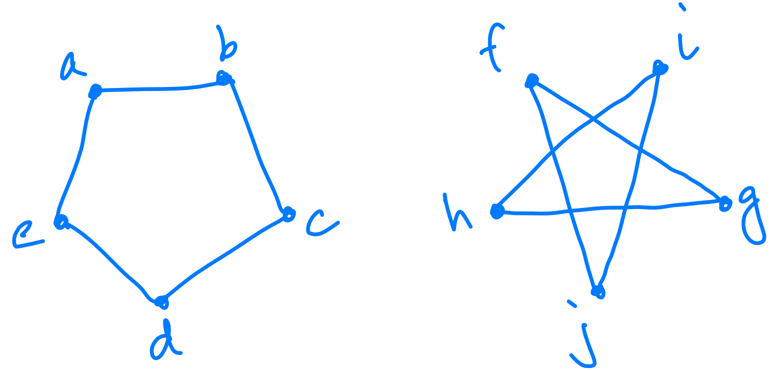
the best algorithms for determining if two graphs are isomorphic have exponential worst case runtime complexity — however, the graph isomorphism problem is not in NP-complete.

* 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 w/ vertices < 100 for isomorphism in < 1 second

All graph properties we care about are preserved under isomorphism!

e.g.



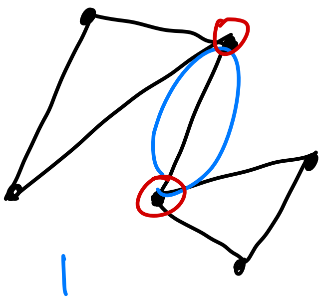
are structurally / semantically equivalent where all graph properties and algorithms are concerned.

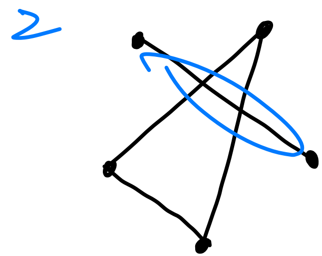
Connectivity

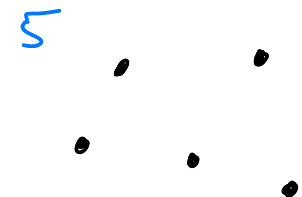
- a graph is connected if there is a path between every pair of vertices
- a connected component of a graph is a subgraph consisting of some vertex and all vertices and edges connected to it.

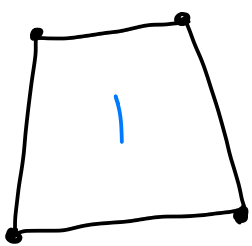
  ∴ a graph is connected iff it has a single connected component.

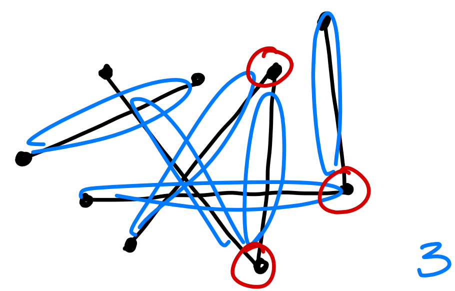
- a cut vertex / cut edge (aka. articulation point / bridge), when removed from a graph, leaves more connected components than before (it disconnects the graph) e.g. for each graph, determine the number of connected components and identify cut vertices/edges, if they exist.











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

kappa

for complete graphs of size n — denoted K_n, define $\kappa(K_1) = 0$, $\kappa(K_n) = n-1$

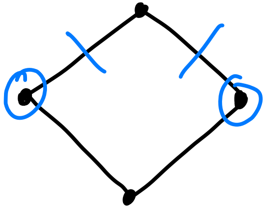
a graph G is k-edge-connected if removing any fewer than k edges does not disconnect it

— the edge connectivity $\lambda(G)$ is the max k s.t. G is k-edge-connected

lambda

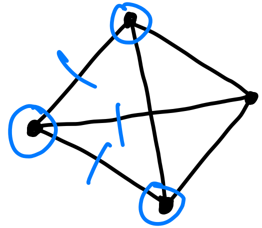
e.g. determine $\kappa(G)$ and $\lambda(G)$ for each graph below

(e.g., these are preserved under isomorphism)



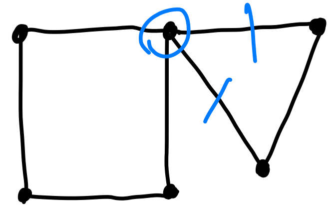
$\kappa(G) = 2$

$\lambda(G) = 2$



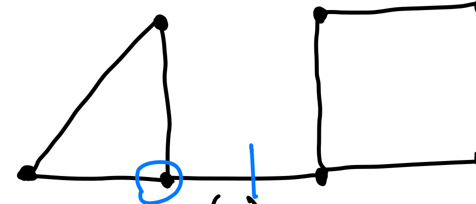
$\kappa(G) = 3$

$\lambda(G) = 3$



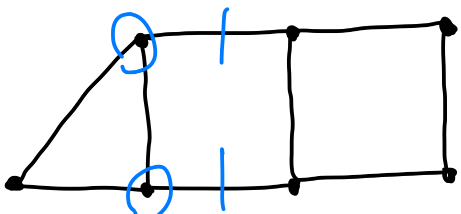
$\kappa(G) = 1$

$\lambda(G) = 2$



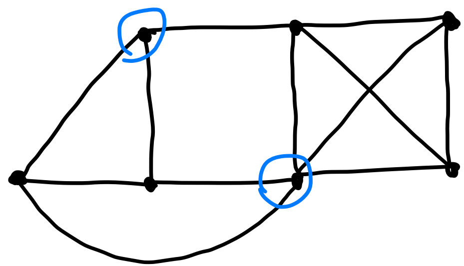
$\kappa(G) = 1$

$\lambda(G) = 1$



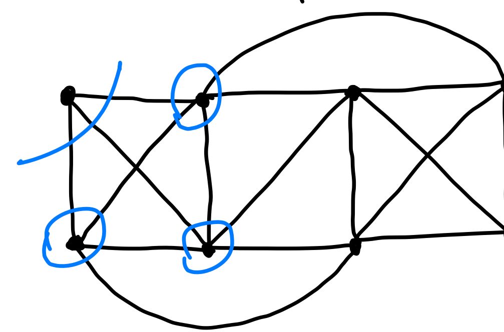
$\kappa(G) = 2$

$\lambda(G) = 2$



$\kappa(G) = 2$

$\lambda(G) = 3$



$\kappa(G) = 3$

$\lambda(G) = 3$

Graph Traversals

"walk" elsewhere

- 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 $e_i = \langle v_{i-1} - v_i \rangle$

"closed walk" elsewhere

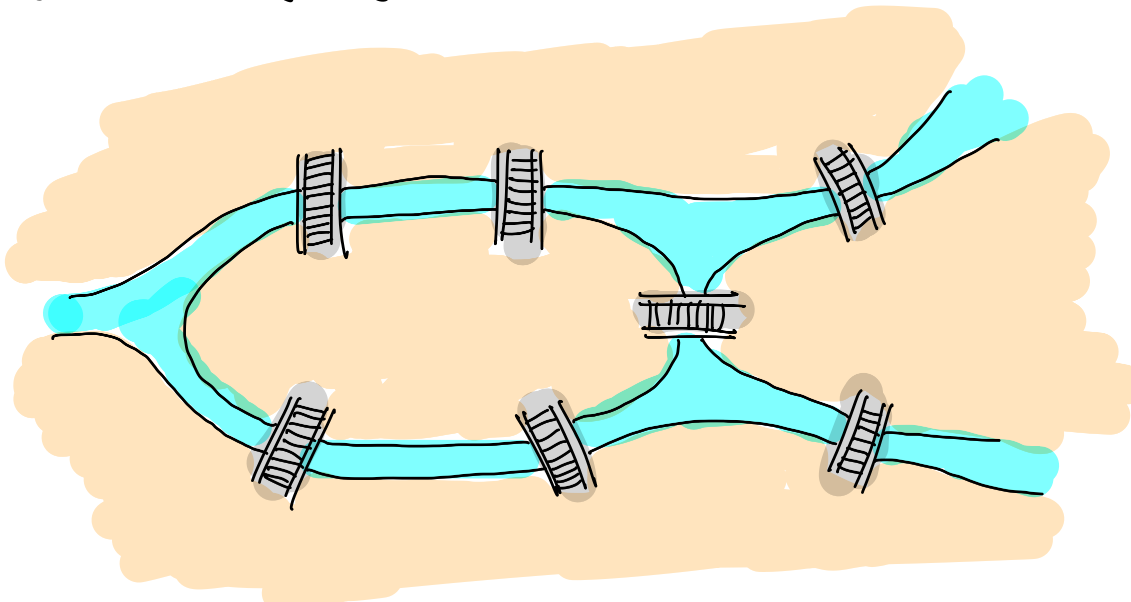
- a circuit is a path that starts and ends on the same vertex

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

"trail/circuit" elsewhere

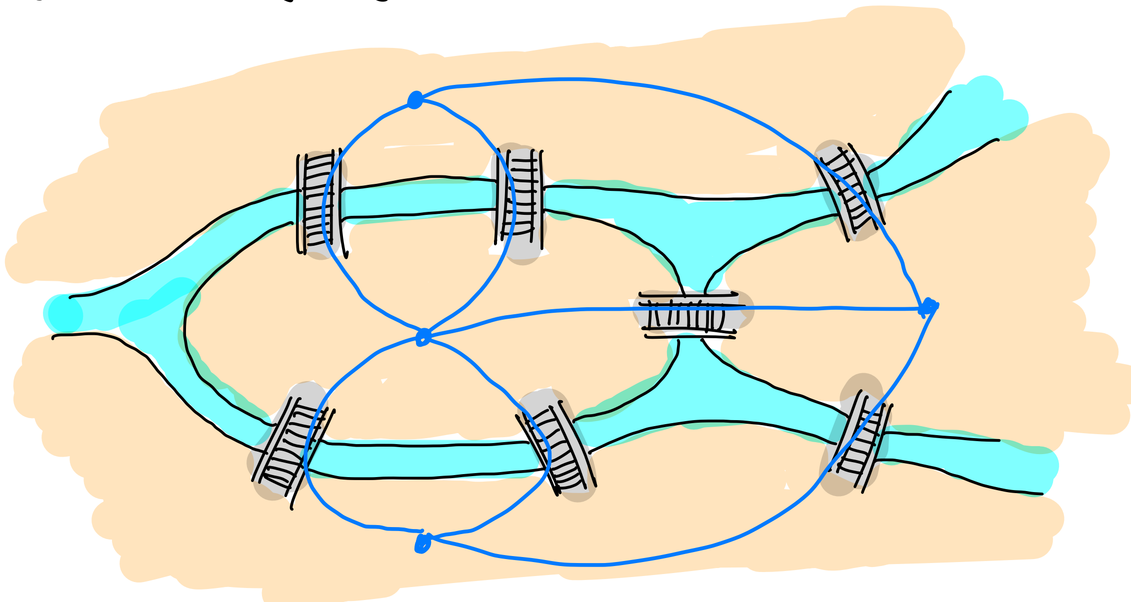
(also: elsewhere "path" = no repeated edges/vertices, and "cycle" = circuit w/ no repeated edges/vertices)

"Bridges of Königsberg" (1736)



Euler: "can we start at some location, traverse all bridges once, and end up where we started?"

"Bridges of Königsberg" (1736)



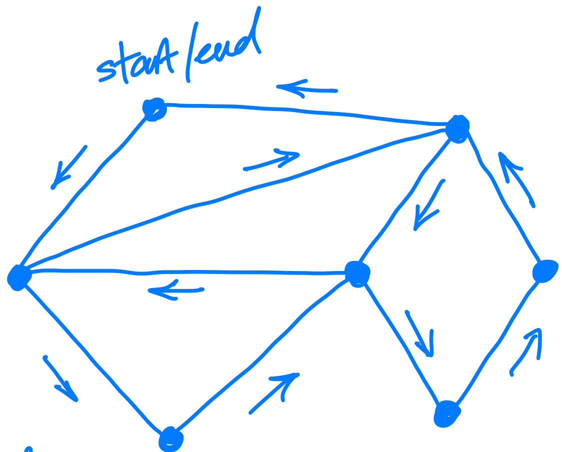
Euler: "can we start at some location, traverse all bridges once, and end up where we started?"

for graph $G = (V, E)$:

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



ok to revisit a vertex, but if isn't the last one, we must leave after visiting.

degrees of all vertices are <u>even</u>

— <u>necessary</u> condition!

also <u>sufficient</u>

- for every node other than start, we must leave after entering
- for start node, we must re-enter + not leave at some point (but can revisit as needed)

(what about Euler paths?)

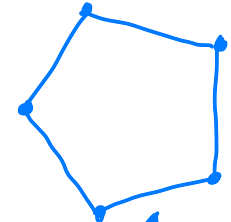
for graph $G = (V, E)$:

- 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: but not necessary!

e.g.



neither holds true, but Ham. circuit exists!

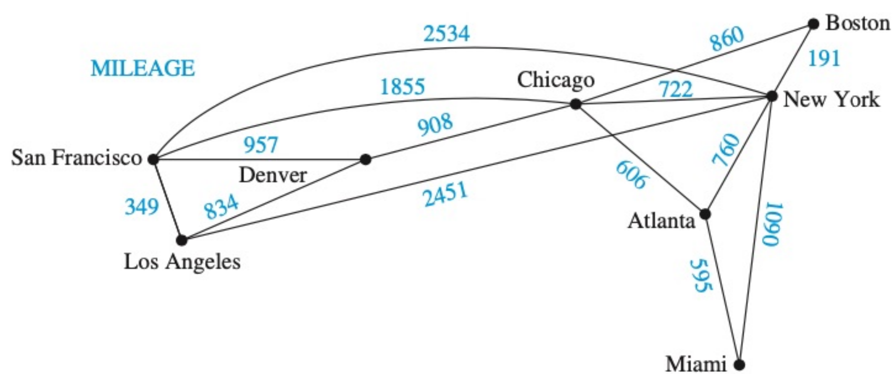
given $G=(V,E)$,

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

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

— no known necessary conditions for Hamiltonian circuits/paths.

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.



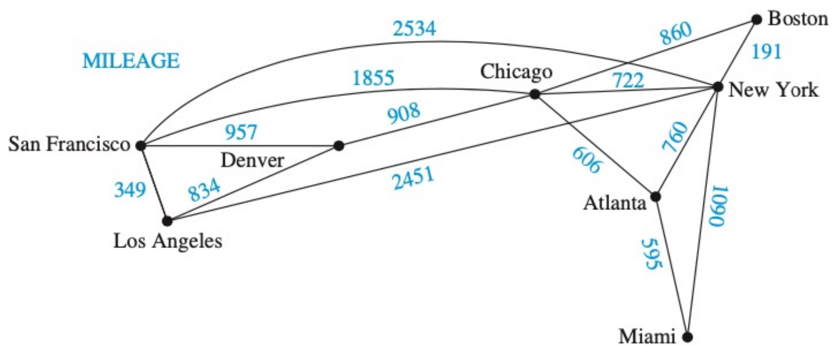
Both the Hamiltonian Circuit Problem (HCP) and TSP are NP-hard problems.

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

$\frac{(n-1)!}{2}$ distinct circuits — 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 w/in 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

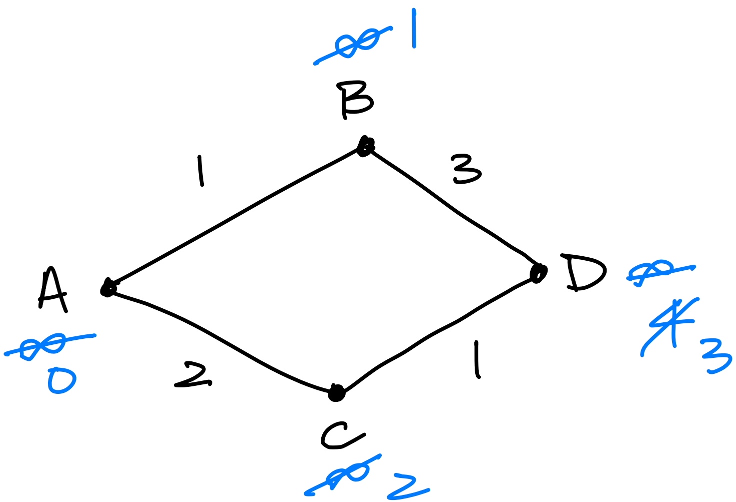
→ ...

Dijkstra's algorithm is an algorithm for finding the shortest path distances in graph $G=(V,E)$ w/ weights $w(u,v) > 0$ for all edges $\langle u\text{-}v \rangle \in E$, from some start node t:

```
S ← ∅
for all u ∈ V:
    L(u) ← ∞
L(t) ← 0
while S ≠ V:
    find node u ∉ S w/ minimum L(u)
    S ← S ∪ {u}
    for all nodes v ∉ S adjacent to u:
        relax(u, v)
```

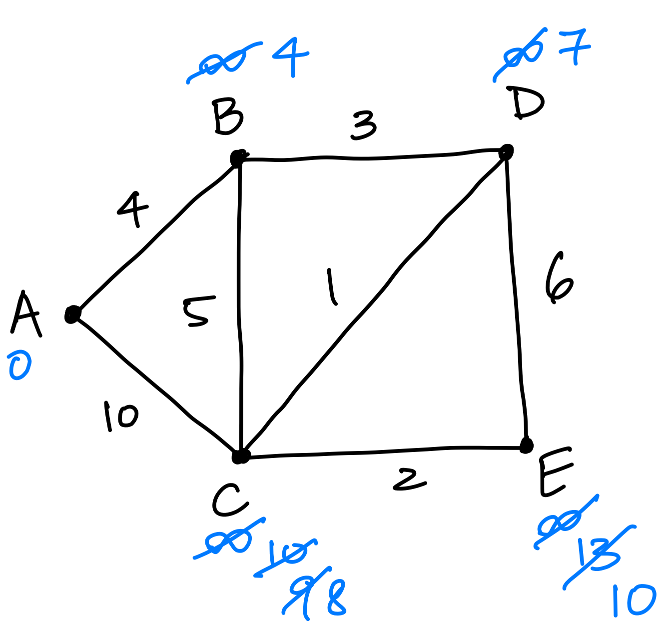
```
relax(u, v):
    if L(u) + w(u,v) < L(v):
        L(v) = L(u) + w(u,v)
```

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 $u \notin S$ w/ minimum $L(u)$
  $S \leftarrow S \cup \{u\}$
  for all nodes $v \notin S$ adjacent to u:
    relax(u, v)

$\in \Theta(|V|^2)$

can be optimized to $\Theta(|E| + |V| \log |V|)$

$|V|$ iterations

at most $|V|-1$ (can be optimized using, e.g., heap)

at most $|V|-1$ (bounded by $|E|$)

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 $G = (V, E)$:

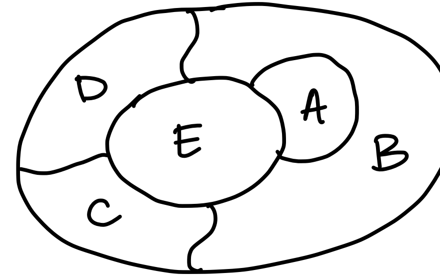
1. choose some ordering of vertices in V:

   $v_1, v_2, \ldots, v_n$

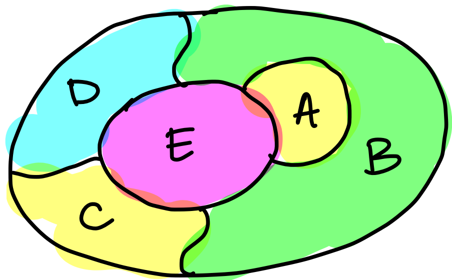
2. order the colors: $c_1, c_2, \ldots$

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)

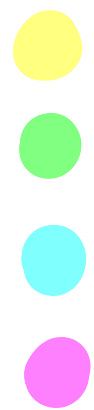
e.g., apply the basic coloring algorithm to this map:



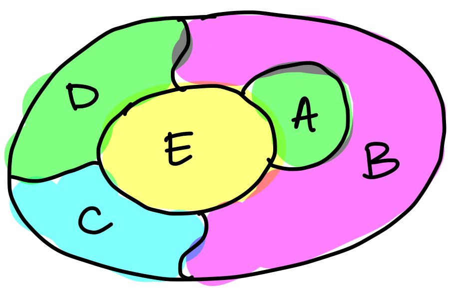
e.g. A B C D E

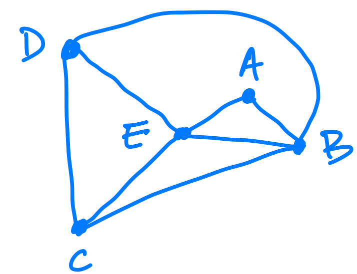


colors:

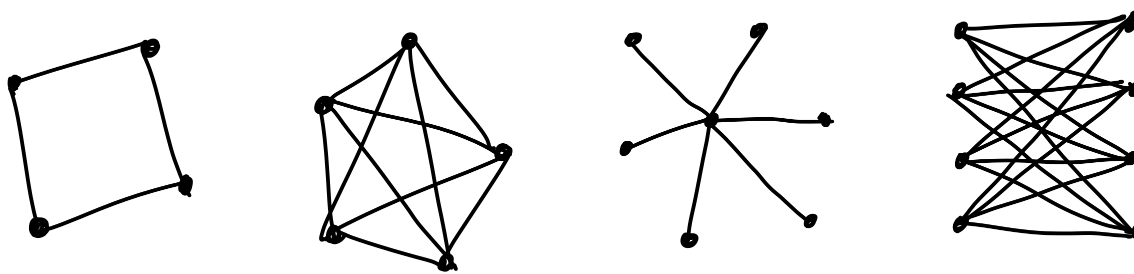


e.g. E D C B A





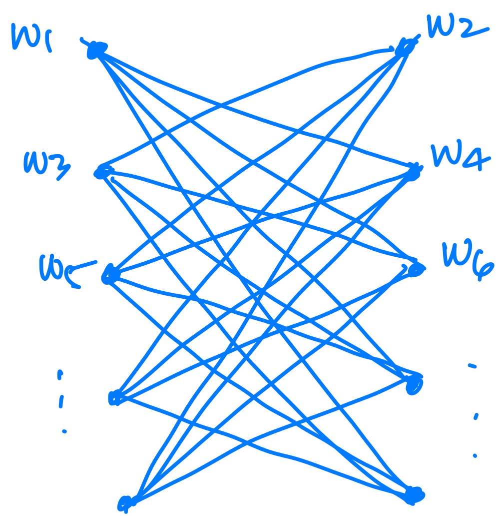
note: we can draw a planar graph representation of any map e.g., how many colors, at most, does the basic coloring algorithm assign to each of the following graphs?



2 5 2 2

– 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 $\chi(G)$.



$\frac{|V|}{2}$ colors assigned,

only 2 needed!

conjecture: given graph $G = (V, E)$ where $|V| = n$ and $\forall w \in V$, $deg(w) \leq k$,
the basic coloring algorithm assigns at most $k+1$ colors to G

PMI: (show $P(1) \wedge P(n) \rightarrow P(n+1)$) — what is predicate P?

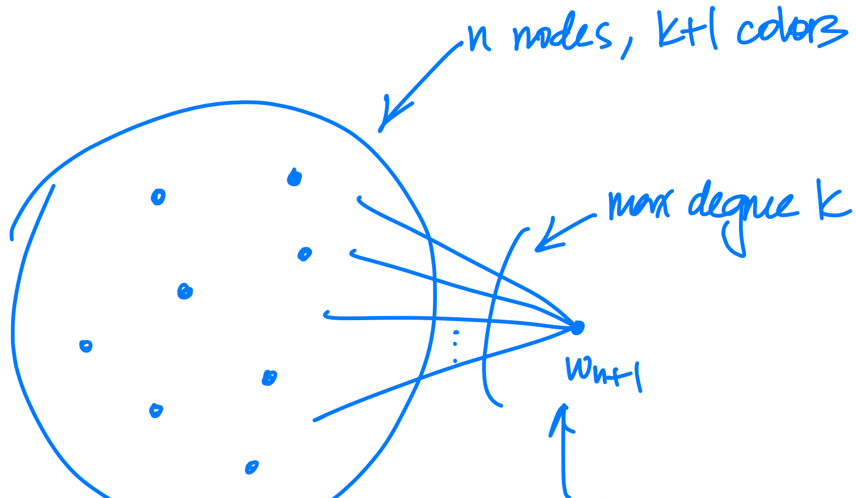
basis: $P(1)$ = graph w/ $|V| = 1$: $\forall w \in V$, $deg(w) = 0$, need $0+1$ colors ✓

inductive step: assume $P(n)$ is true (I.H.)

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

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

by I.H., we know that this subgraph require at most $k+1$ colors i.e.,



because this node has at most k neighbors (w/ k distinct colors), $\exists$ color $\in \{c_1, c_2, \ldots c_{k+1}\}$ that we can assign to w_{n+1}

$\therefore P(n) \rightarrow P(n+1)$ ✓

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 $k > 2$ is NP-complete

(except for some special cases —

e.g., 2-coloring — $O(|V| + |E|)$

$k > 3$ for planar graphs)