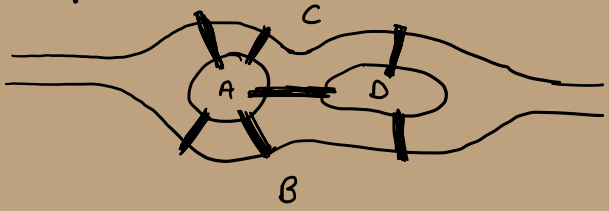
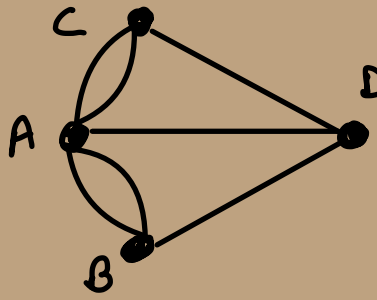
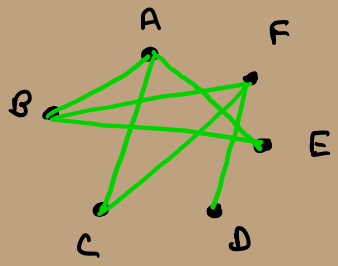
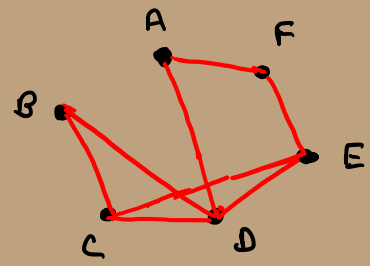




Graphs

Theory, Models, Algorithms





John Rachlin

CS5002 / CS1800
Discrete Structures

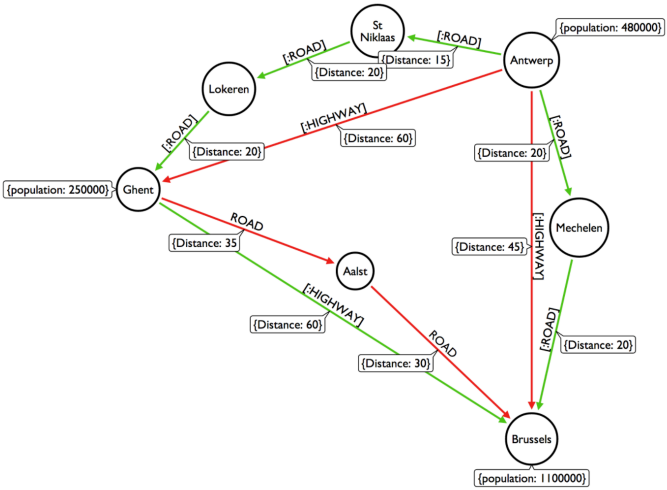
Northeastern Univ.

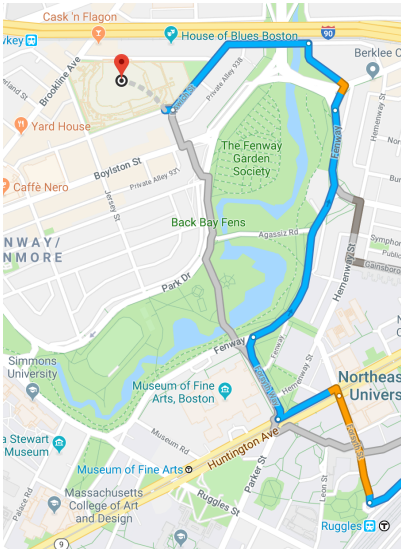
CS5002/CS1800:
Discrete Structures

Graphs

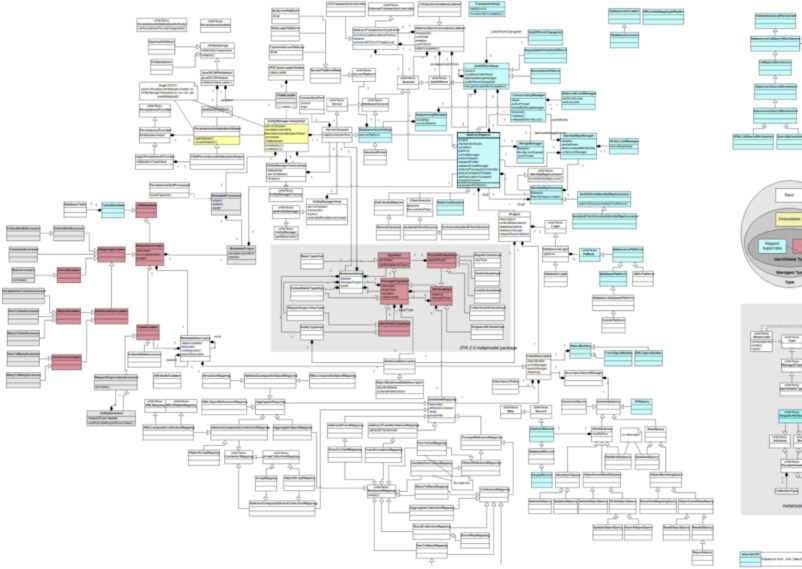
Theory, Models, and Algorithms

Graphs are Ubiquitous…in Navigation and Routing

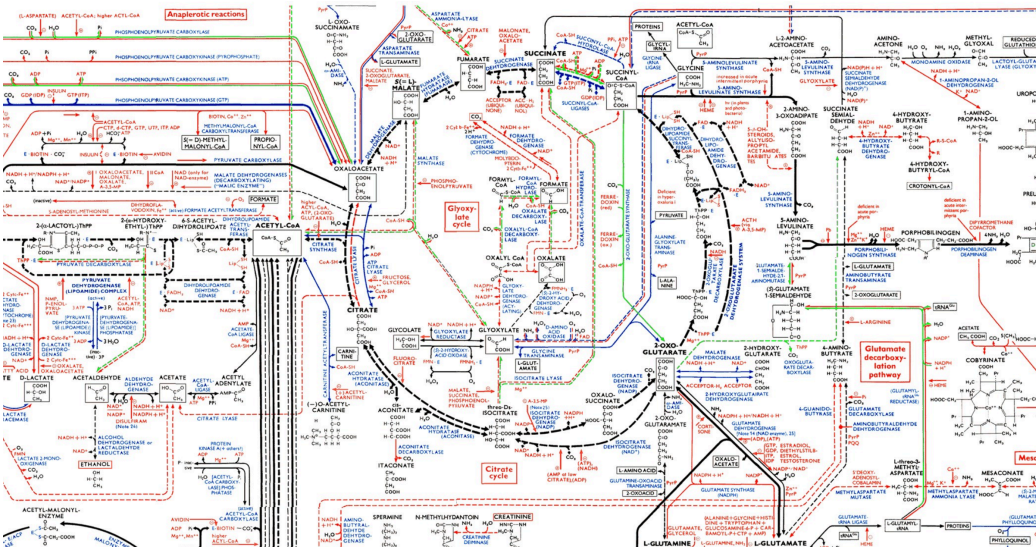




Graphs are Ubiquitous…in software design



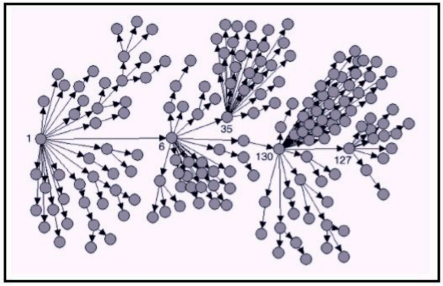
Graphs are Ubiquitous…In Biology



Graphs are Ubiquitous…in medicine

Is there a small subset of SARS patients who account for a disproportionate share of transmission? **MMWR May 9, 2003 / Vol. 52 / No. 18 - I**

FIGURE 2. Probable cases of severe acute respiratory syndrome, by reported source of infection* — Singapore, February 25–April 30, 2003



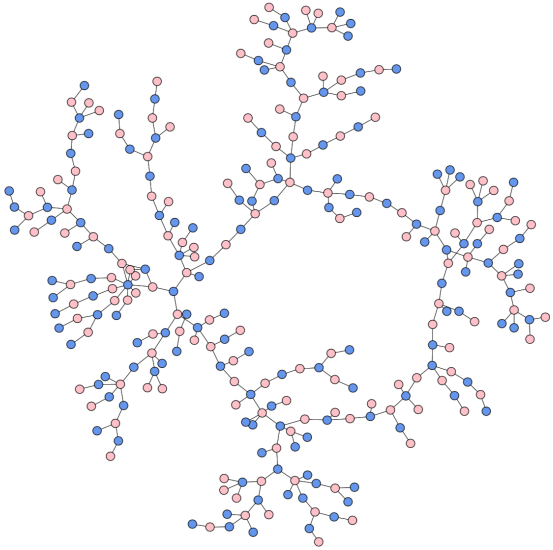
* Patient 1 represents Case 1; Patient 6, Case 2; Patient 35, Case 3; Patient 130, Case 4; and Patient 127, Case 5. Excludes 22 cases with either no or poorly defined direct contacts or who were cases translocated to Singapore and the seven contacts of one of these cases.

Reference: Bogatti SP. Netdraw 1.0 Network Visualization Software. Harvard, Massachusetts: Analytic Technologies, 2002.

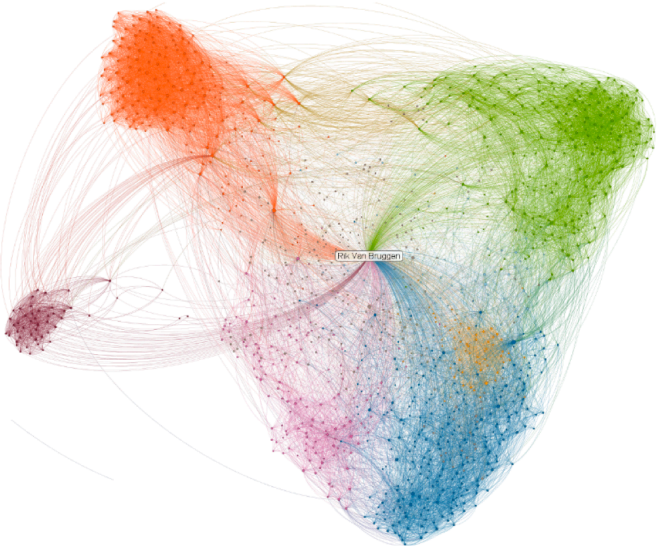
Patients No: 1, 6, 35, 130&127 seemed to be "**hypertransmitters**"

Graphs are Ubiquitous…Social Networks

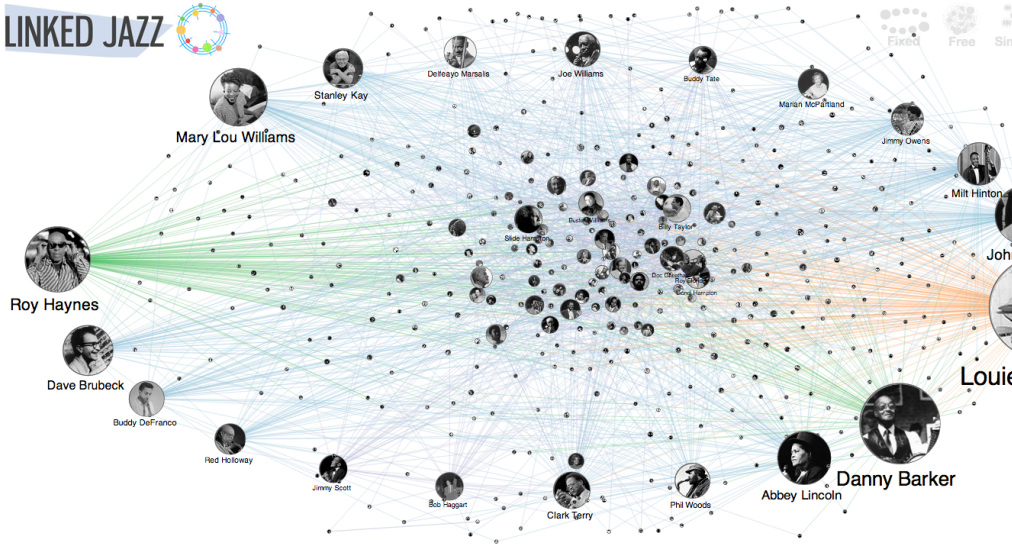
High school
Dating Network



Graphs are Ubiquitous…in society



Graphs are Ubiquitous…in music



Graphs are Ubiquitous…in literature

TITUS ANDRONICUS

Number of characters 36 | 50% Network density

ROMEO AND JULIET

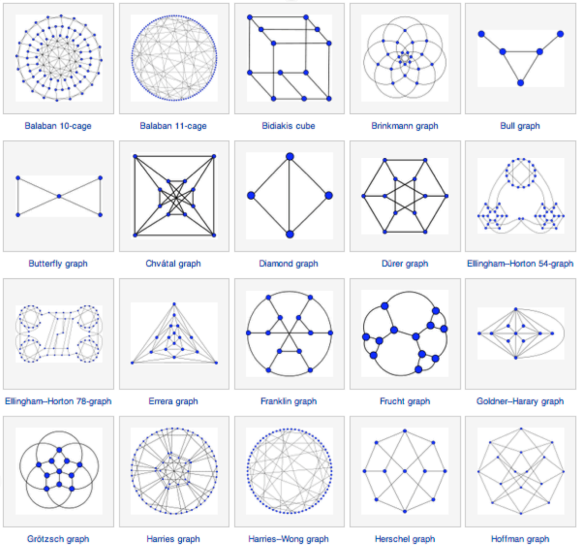
Number of characters 41 | 37% Network density

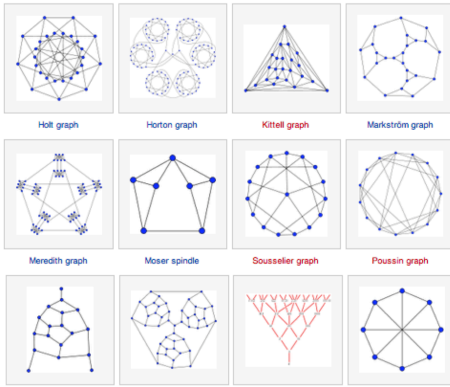
JULIUS CAES

Number of characters 46 | 34%

From: www.martingrandjean.ch/network-visualization-shakespeare

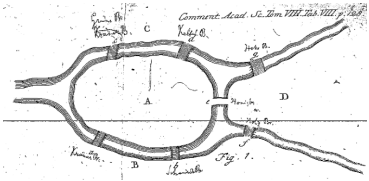
Graph Types

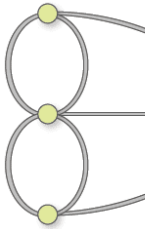

Balaban 10-cage
Balaban 11-cage
Bidiakis cube
Brinkmann graph
Bull graph
Butterfly graph
Chvátal graph
Diamond graph
Dürer graph
Ellingham–Horton 54-graph
Ellingham–Horton 78-graph
Errera graph
Franklin graph
Frucht graph
Goldner–Harary graph
Grötzsch graph
Harries graph
Harries–Wong graph
Herschel graph
Hoffman graph


Holt graph
Horton graph
Kittell graph
Markström graph
Meredith graph
Moser spindle
Sousselier graph
Poussin graph
Tutte's fragment
Tutte graph
Young–Fibonacci graph
Wagner graph

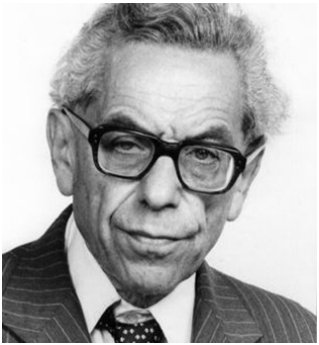
Great names in Graph Theory

Leonhard Euler
1707 - 1783



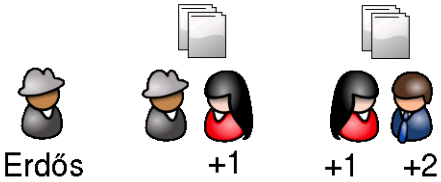


Great names in Graph Theory



Paul Erdős
1913 - 1996

Erdős Numbers



Albert Einstein: 2
Richard Feynman: 3
Stephen Hawking: 4

JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 13, Number 10, 2006

Pp. 1659–1672

Multi-Node Graphs: A Framework for Multiplexed Biological Assays

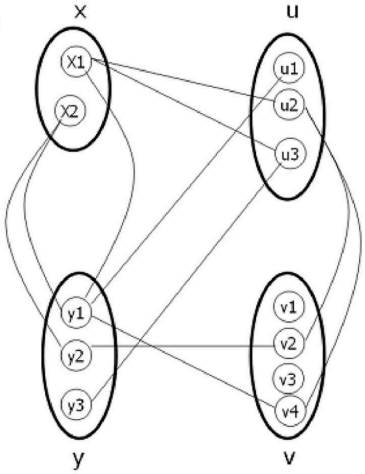
NOGA ALON,[1,2] VERA ASODI,[2] CHARLES CANTOR,[3] SIMON KASIF,[3,4] and JOHN RACHLIN[5]

ABSTRACT

Multiplex polymerase chain reaction (PCR) is an extension of the standard PCR protocol in which primers for multiple DNA loci are pooled together within a single reaction tube, enabling simultaneous sequence amplification, thus reducing costs and saving time. Potential cost saving and throughput improvements directly depend on the level of multiplexing achieved. Designing reliable and highly multiplexed assays is challenging because primers that are pooled together in a single reaction tube may cross-hybridize, though this can be addressed either by modifying the choice of primers for one or more amplicons, or by altering the way in which DNA loci are partitioned into separate reaction tubes. In this paper, we introduce a new graph formalism called a *multi-node graph*, and describe its application to the analysis of multiplex PCR scalability. We show, using random multi-node graphs that the scalability of multiplex PCR is constrained by a phase transition, suggesting fundamental limits on efforts to improve the cost-effectiveness and throughput of standard multiplex PCR assays. In particular, we show that when the multiplexing level of the reaction tubes is roughly $\Theta(\log(sn))$ (where s is the number of primer pair candidates per locus and n is the number of loci to be amplified), then with very high probability we can 'cover' all loci with a valid assignment to one of the tubes in the assay. However, when the multiplexing level of the tube exceeds these bounds, there is no possible cover and moreover the size of the cover drops dramatically. Simulations using a simple greedy algorithm on real DNA data also confirm the presence of this phase transition. Our theoretical results suggest, however, that the resulting phase transition is a fundamental characteristic of the problem, implying intrinsic limits on the development of future assay design algorithms.

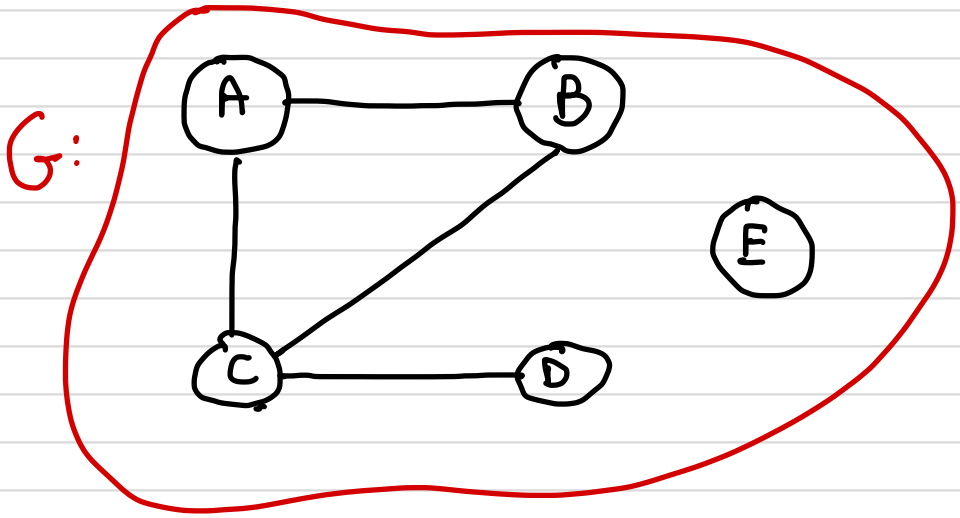
Key words: algorithms, combinatorics, computational molecular biology.

(a)



(b)

A graph $G = (V, E)$ is a non-empty set of vertices or "nodes", V and a set of edges, E. Each edge is associated with one or two vertices or end points.

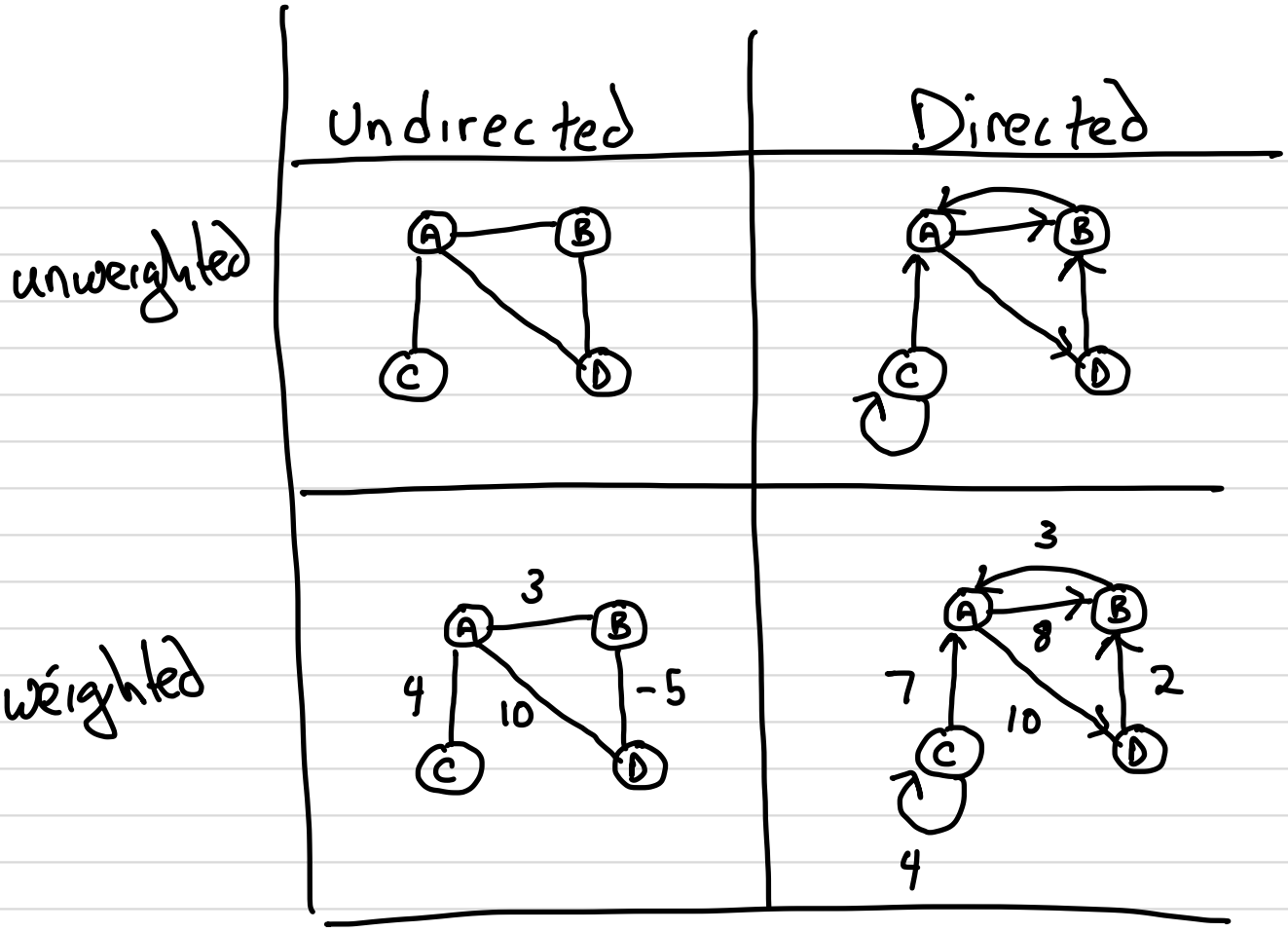


$$V = \{A, B, C, D, E\}$$

$$E = \{\{A,B\}, \{A,C\}, \{B,C\}, \{C,D\}\}$$

$$|V| = 5$$

$$|E| = 4$$

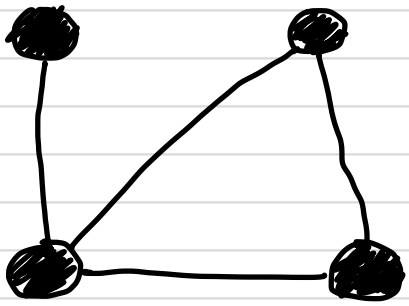


Directed Graphs: The edges have a from-node / out-node and a to-node / in-node. Loops are ok.

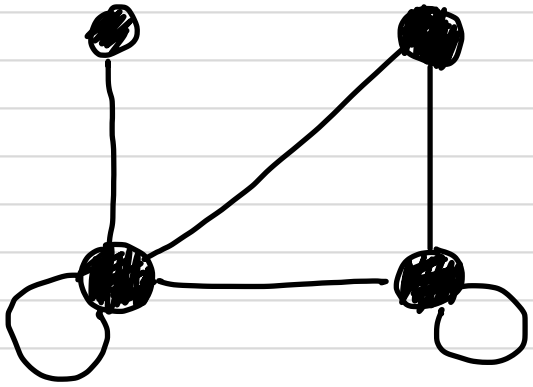
Weighted Graph: A number is attached to each edge. What the number represents depends on the problem (distance, cost, capacity, etc.)

A simple graph is an unweighted, undirected graph w/ at most one edge between any two vertices and no loops.

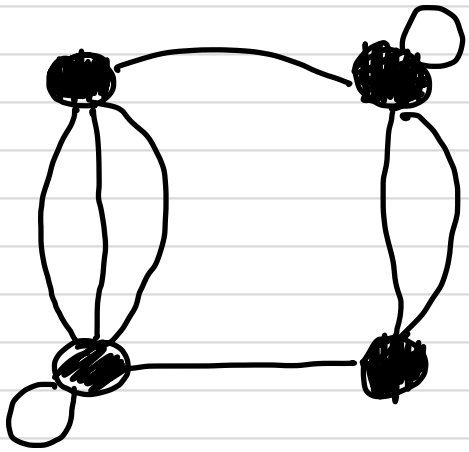
simple



non-simple (loops)

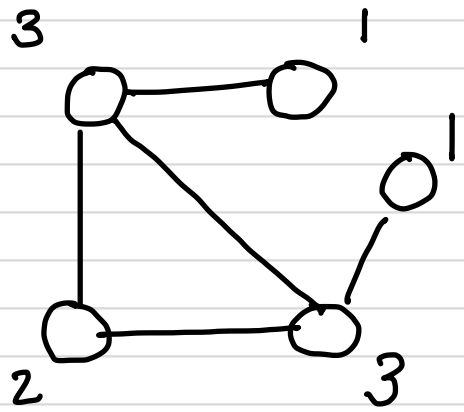


non-simple (loops and multi-edges)



Definitions

The degree of a vertex, $\deg(v)$, is the # of edges connected to the vertex.



For directed graphs, we distinguish in-degree, out-degree

↑ # incoming edges

↑ # outgoing edges

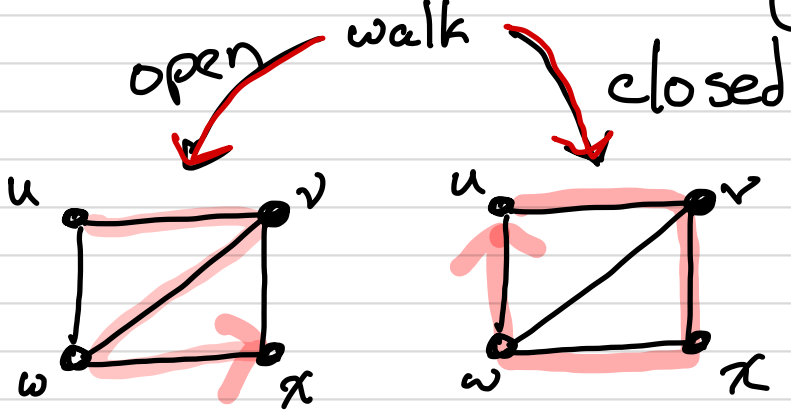
Two vertices are adjacent if they are connected by an edge. Two edges are adjacent if they share a vertex.

A walk, $u \rightsquigarrow v$, is a sequence of adj. edges from u to v.

Note: Some texts refer to this as a "path".

Definitions

walk: Traversing from one vertex to another along edges.



A walk from $u \rightsquigarrow x$ (open)

A walk from $u \rightsquigarrow u$ (closed)

↓

trail

open walk with no repeating edge

↓

path

trail with no repeating vertex

↓

circuit

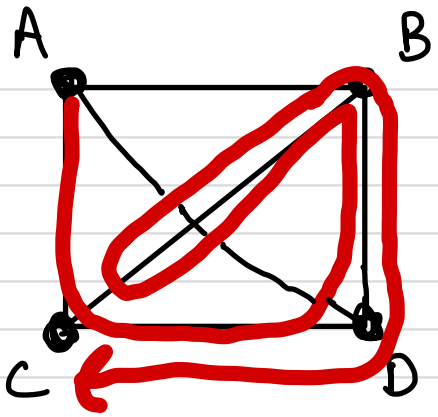
closed walk with no repeating edge

↓

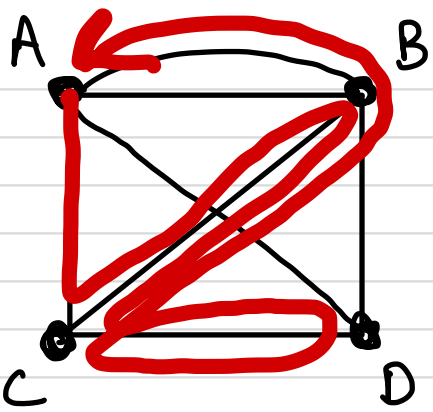
cycle

circuit w/ no repeating vertex.

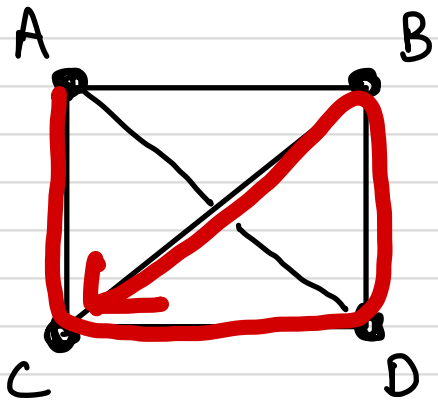
open walk



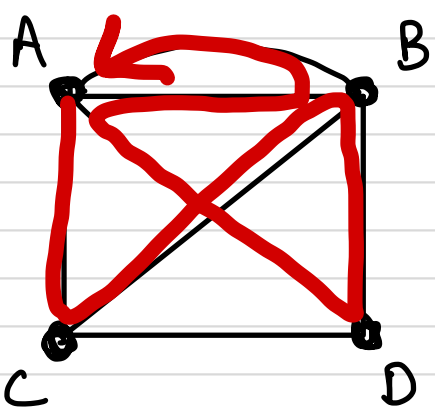
closed walk



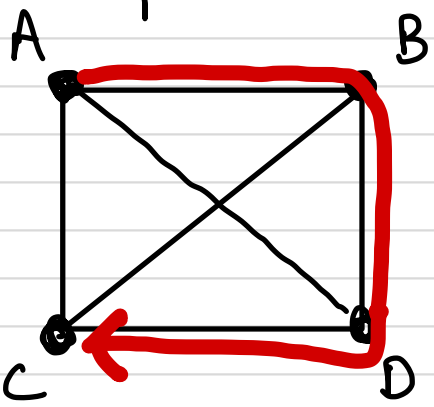
trail



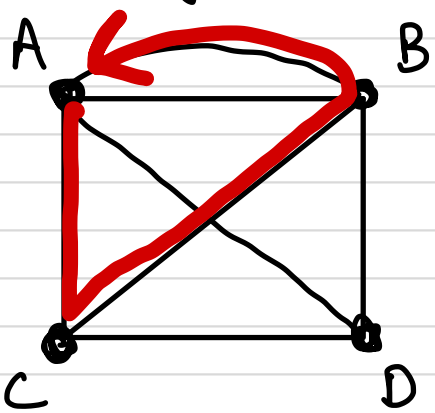
circuit



path



cycle

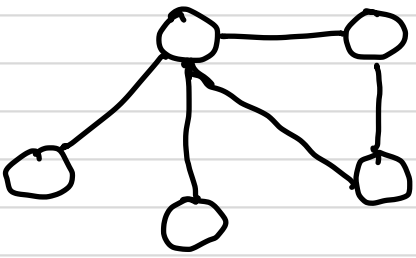


Reachability

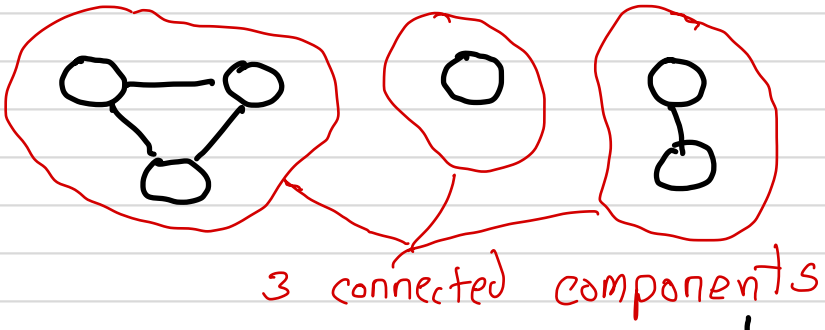
u is reachable from v if there is a walk from $u \rightsquigarrow v$

Connected / Unconnected Graphs

A connected (undirected) graph is a graph where there are paths between any pair of vertices.



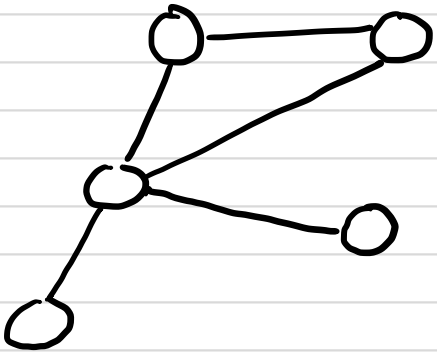
connected



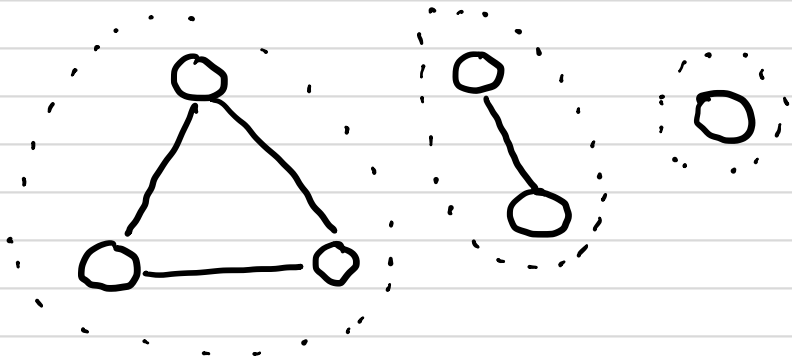
un connected

Concepts - continued.

A Graph is connected or it isn't.



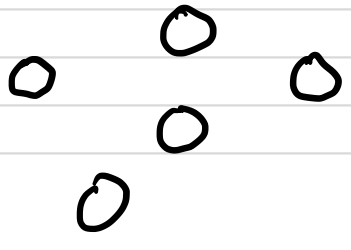
Connected



Not Connected.
But it has 3 connected components

- A connected graph has one connected component.
- A connected graph has a path between any arbitrary pair of vertices: $x \overset{p}{\rightsquigarrow} y$
- There are no paths or edges between vertices in different connected components only within the same connected component.

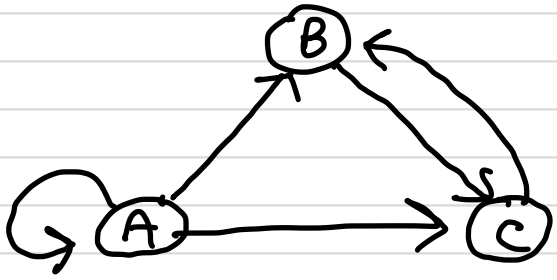
- A graph could have no edges. (Its not very useful however)



(Unconnected doesn't mean no edges, it only means there are 2 or more connected components!)

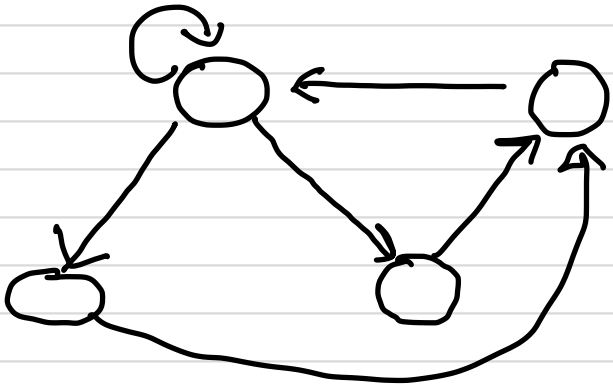
Connectivity in Directed Graphs

weakly connected



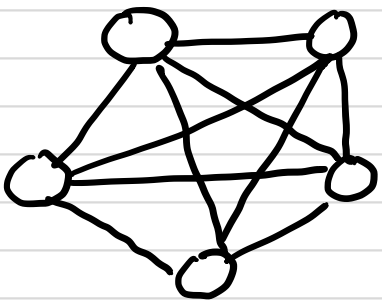
(no path $C \rightsquigarrow A$)

strongly connected



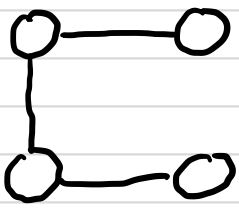
There is a directed path between any two vertices.

A <u>complete</u> graph means there is an actual edge between every pair of vertices.



Complete $\Rightarrow$ Connected

Connected $\not\Rightarrow$ Complete

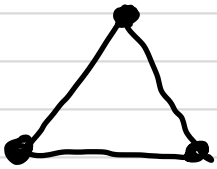


- connected
- complete

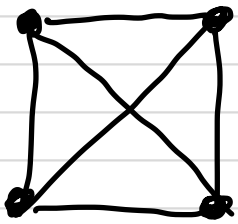
- connected
- not complete

$|V| = 1$
$|E| = 0$

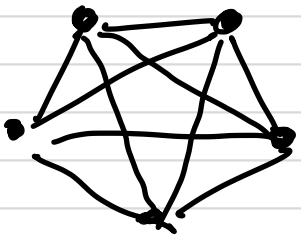
$|V| = 2$
$|E| = 1$



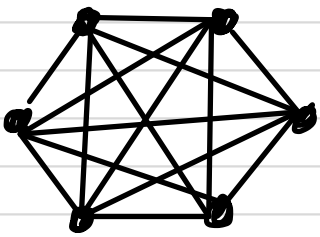
$|V| = 3$
$|E| = 3$



$|V| = 4$
$|E| = 6$



$|V| = 5$
$|E| = 10$

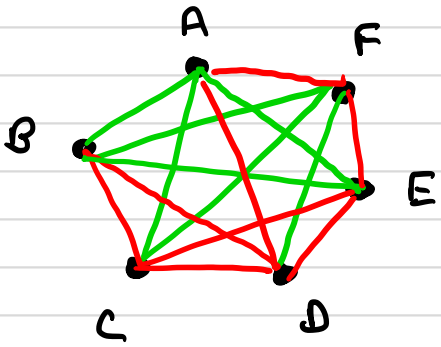


$|V| = 6$
$|E| = 15$

$$|E| = \binom{|V|}{2} = \frac{|V| \cdot (|V| - 1)}{2}$$

Friends and Strangers

In a group of 6 people, there is guaranteed to be three people who are mutual friends or three people who are mutual strangers.



friends

strangers

Proof: Every vertex has degree 5 - So 5 edges to either friends or strangers. By the Pigeon hole principle, every vertex has either 3 or more friends OR three or more strangers.

Without loss of Generality, consider A.
"We could apply the following argument to B, C, ... F."

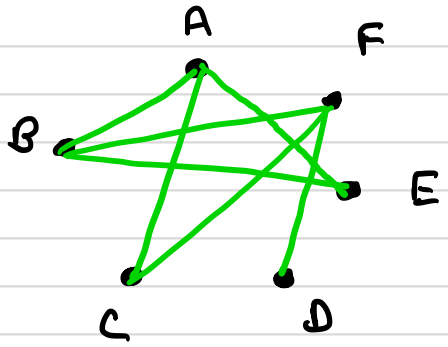
A has three friends: B, C, E.

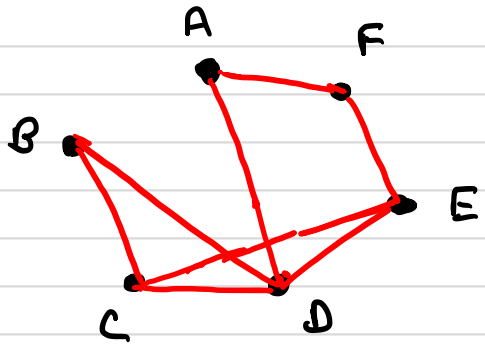
Case I: There is at least one friend among BCE. For A, this case applies! BE are friends. So this forms a triangle $\triangle_{ABE}$ that are mutual friends

Case II: (Doesn't apply to A but does apply to F): If BCE had no mutual friends then $\triangle_{BCE}$ would be a group of mutual strangers

For C (which has 3 strangers) we reverse the argument. ∎

Cliques, Independent Sets, Graph Complements





F: Friend Graph $= \bar{S}$ S: Stranger Graph $= \bar{F}$

F and S are <u>complementary</u>: An edge in F means no edge in S and vice versa.

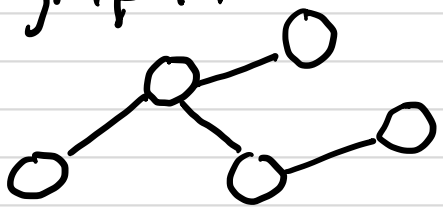
In F, ABE are mutual friends. They form a <u>clique</u> (pronounced "kleek") (i.e., a complete <u>subgraph</u>)

In S, ABE are an <u>independent set</u>. There are <u>no</u> edges connecting A, B, E.

Do you see the connection between cliques, independent sets, and graph complements?

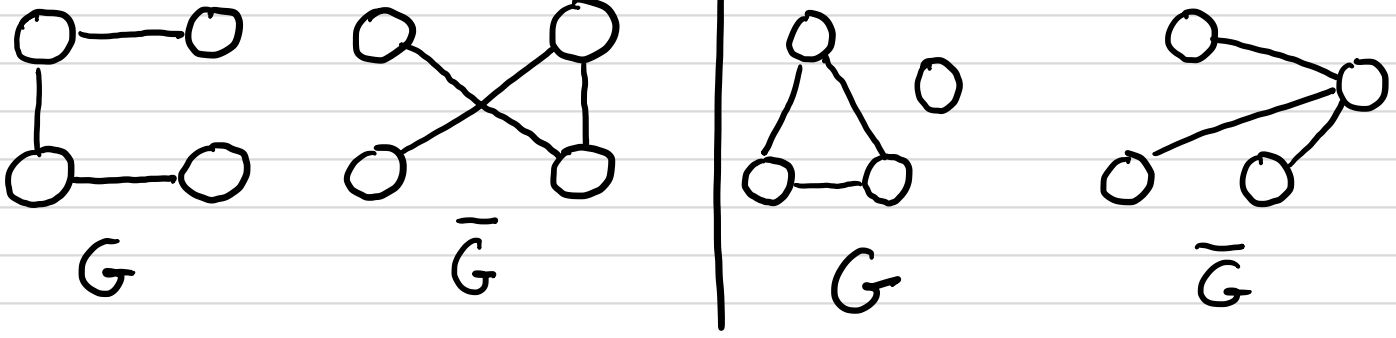
Trees are a special kind of graph

- minimally connected (removal of any edge disconnects the graph.
- no cycles
- $|V| = |E| + 1$

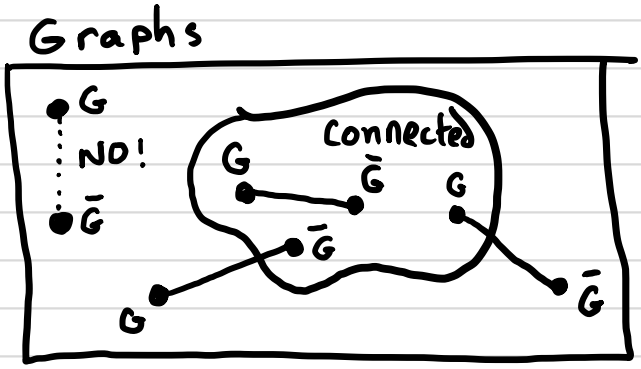


(more about trees soon.)

A complement of a graph, $\bar{G}$, has an edge wherever G doesnt and vice versa.



Prove: G or $\bar{G}$ is connected. (Or both!)



Note: This can also be proved by mathematical induction:

$\forall n: G_n$ or $\bar{G}_n$ is connected where $n = |V|$

Proof:

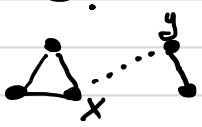
G is connected or not connected

Case I: G is connected. Statement true (trivially)

Case II: G not connected

1. G has 2 or more connected components
2. Pick two arbitrary vertices $x \neq y$. They are in same connected component or different connected component in G.

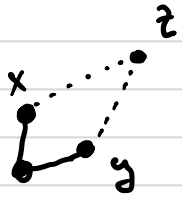
Case A. Different.



$(x,y) \notin E$ in G $\therefore$ $(x,y) \in E$ in $\bar{G}$

$\therefore$ $x \tilde{p} y$ in $\bar{G}$

Case B. Same. Pick any vertex z in a different connected component.
$(x,z) \in E$ in $\bar{G}$, $(y,z) \in E$ in $\bar{G}$
So $x \tilde{p} y$ in $\bar{G}$ (via z)

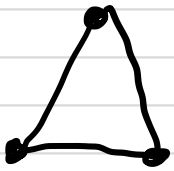


$\therefore$ 3. $x \tilde{p} y$ in $\bar{G}$

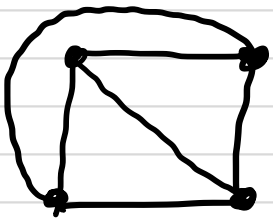
$\therefore$ 4. $\bar{G}$ is connected. ■

Planar Graphs

A <u>planar</u> graph is a graph that can be drawn so that no edge crosses another

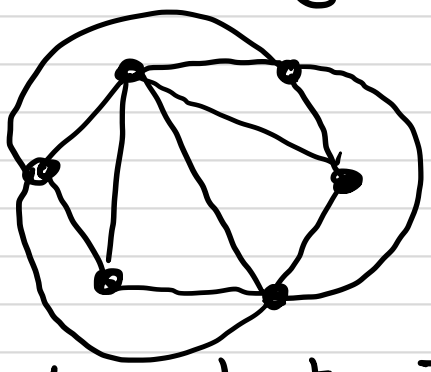


Planar



Planar!

How many edges can you draw for $n = 6$ while keeping the graph planar?

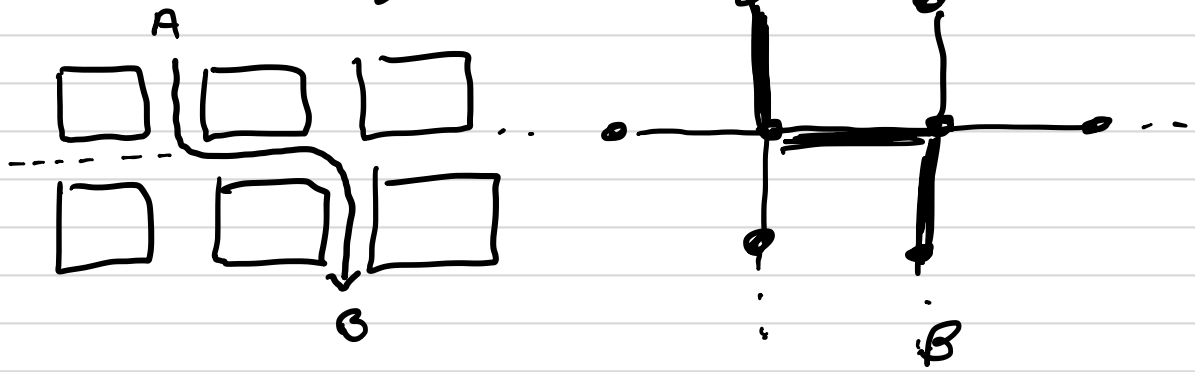


$|V| = 6$

$|E| = 12$

If can be proved that If a graph is planar $|E| \leq 3 \cdot |V| - 6$

Many graphs, like road maps are planar (ignoring bridges and overpasses)



So a map with, say 10^6 intersections:

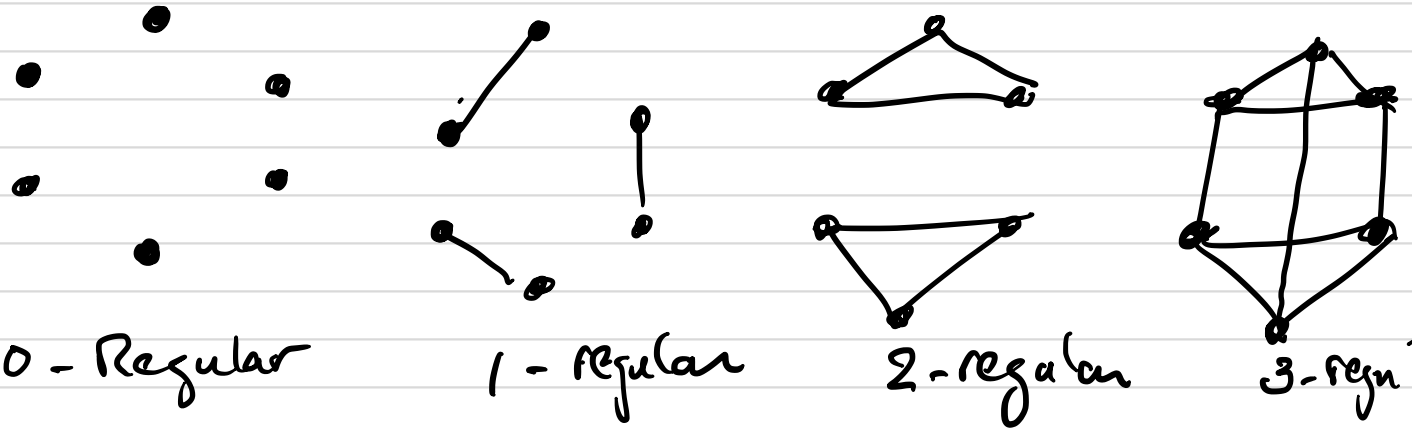
Memory Requirements

Adj. Matrix: $\approx (10^6)^2 = 10^{12} \sim 1$ Terabyte

Adj. List $\approx 10^6 + 2 \cdot (3 \cdot 10^6) \simeq 7$ megabytes. (much better!)

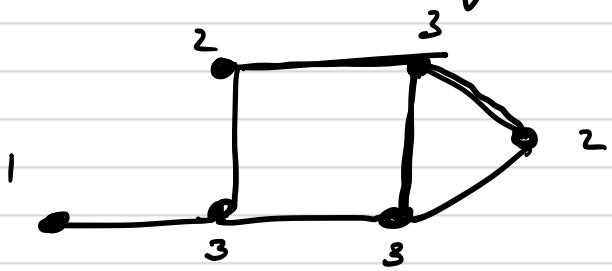
Regular Graphs

A graph is <u>regular</u> if all vertices have the same degree.



0-Regular 1-regular 2-regular 3-regular

Sum - Degree Formula



$$\sum_{v} \deg v = 2 \cdot |E|$$

$14 = 2 \cdot 7$ ✓

Proof: When summing degrees we count each edge exactly twice, once for the current vertex and once for the adjacent vertex.

Handshaking Lemma: The number of vertices with odd degree is even.

"At a party, an even # of people will shake hands with an odd # of people"

$$\sum_{v} \deg v = \sum_{\text{even}} \deg v + \sum_{\text{odd}} \deg v$$

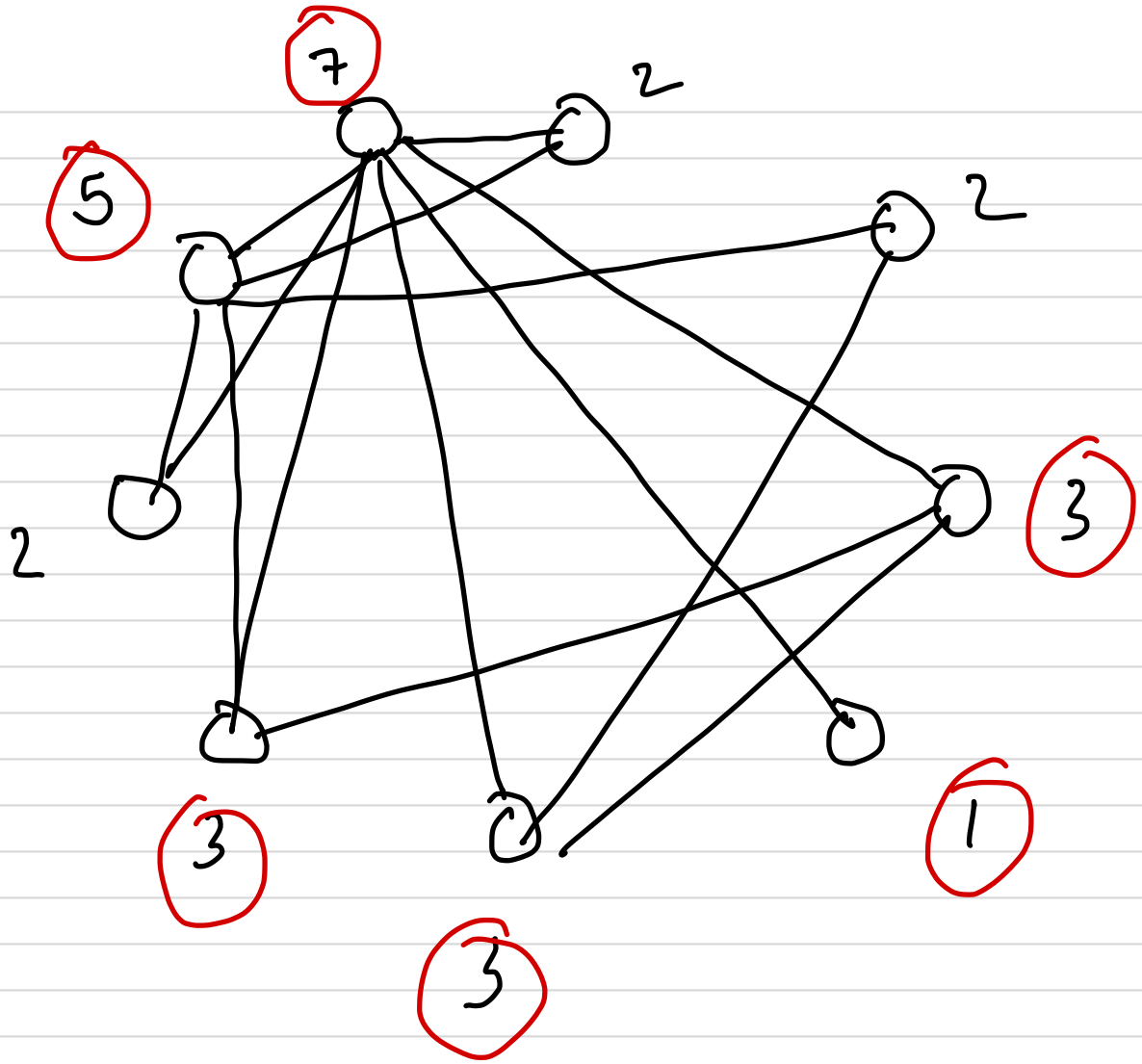
$$\underbrace{2 \cdot |E|}_{\text{even \#}} = \underbrace{\quad}_{\text{even \#}} + \underbrace{\quad}_{\boxed{\text{even \#}}}$$

$2 \cdot |E| = 11$

can't be odd!

In order for $\sum_{\text{odd}} \deg v$ to be even, we must be summing together an even # of odd values.

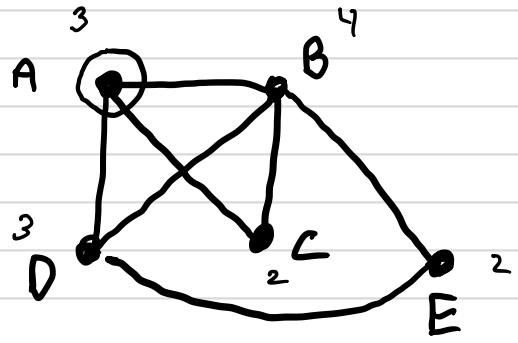
∴ There are an even # of odd-degree vertices.



6 vertices have odd degree

(Pron. "Oiler")

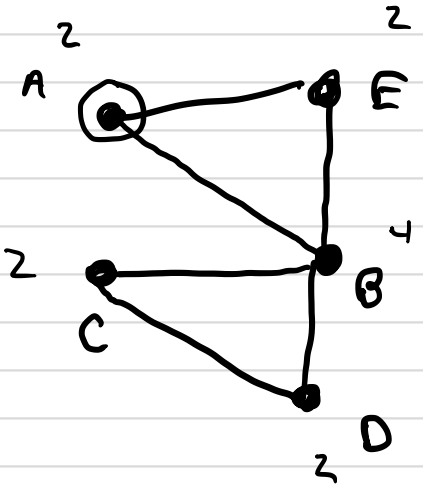
⑧ An <u>Euler Trail</u> is a trail that crosses every edge exactly once. An Euler path is an <u>Euler circuit</u> if the trail begins and ends at the same vertex.



ABCADBED

# Even	3	Euler
# Odd	2	Trail

A, D: The two odd vertices are the start and end of the path!



ABCDBEA Euler Circuit.

# Even Vertices:	5
# Odd Vertices:	0

Euler Proved:

- A graph has an Euler circuit if and only if the degree of every vertex is even.
- A graph has an Euler trail if and only if there are 0 or 2 vertices with odd degree.

The first statement embodies <u>two</u> claims

A) If G has all even vertices then G has an Euler circuit.

B) If G has an Euler circuit, then all vertices have even degree.

Proof strategies:

If P Then Q ← Prove This

Assume P.
$P \Rightarrow R$
$R \Rightarrow S$
$S \Rightarrow T$
$T \Rightarrow Q$ ∎

"Direct Proof"

Assume Q is false.
$\neg Q$
$\neg Q \Rightarrow U$
$U \Rightarrow V$
$V \Rightarrow \neg P$

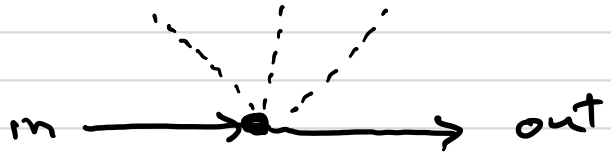
"Contrapositive Proof"

Contrapositive Proof of (B):

Suppose G has an odd degree vertex, v. When a circuit visits v, 2 edges are "used up", (incoming + outgoing) leaving an odd # of edges remaining.

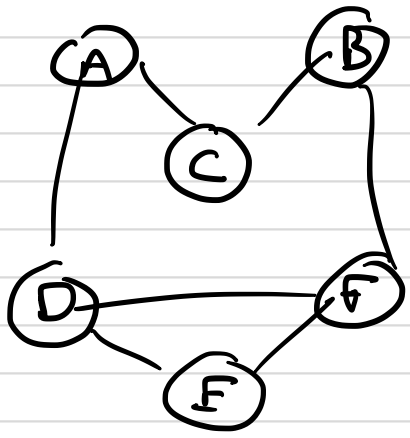
So for v, no circuit leaves zero edges unused.

So the circuit is not an <u>Euler</u> circuit.



Graph Representations

(A) Adjacency Matrix



	A	B	C	D	E	F
A	0	0	1	1	0	0
B	0	0	1	0	0	1
C	1	1	0	0	0	0
D	1	0	0	0	1	1
E	0	0	0	1	0	1
F	0	1	0	1	1	0

a) wasteful : A-C and C-A are both listed (undirected graphs)

b) Memory $\sim |V|^2$ or $O(|V|^2)$

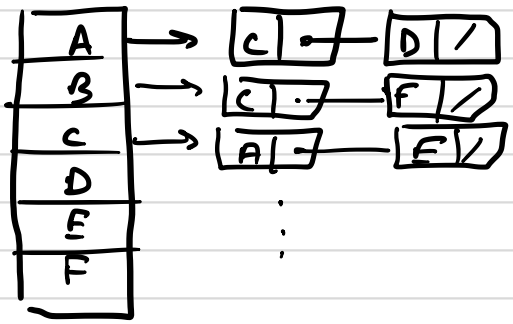
c) Determining adjacency is fast

$O(1)$ - an array look up

d) Weighted graphs are trivial — put weights instead of 0/1

e) O.K. for very small / dense graphs

(B) Adjacency List



} A list of all the things connected to A. We aren't saying that C is connected to D!

$n = |V|$ $\quad 2 \cdot |E|$

a) More memory efficient:

$$\text{Memory} \approx |V| + 2|E|$$

b) for most graphs, $|E| \approx k \cdot |V|$

($|E|$: # edges; $|V|$: # vertices)

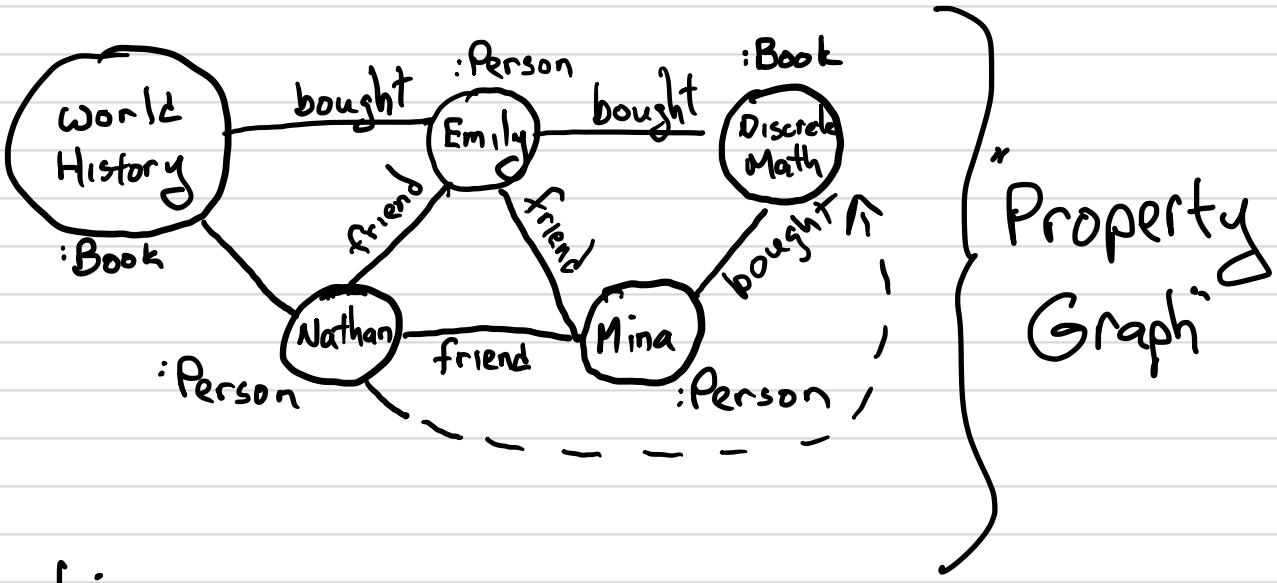
with $k \approx 1 \ldots 3$.

c) Fast retrieval of all adjacent vertices.

d) Best for large / sparse graphs

Graph Databases

Graph Databases are a special type of database for storing graphs and asking questions about their properties



Applications:

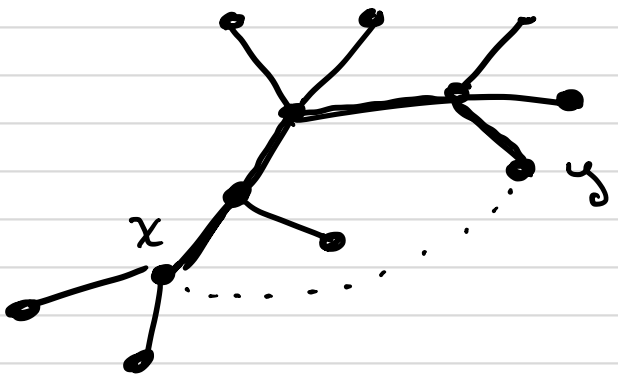
- Recommendation Engines
- Modeling supply chains
- Biological Pathways / Drug Discovery

Q: What book should Amazon recommend to Nathan?

A: Discrete Math!

TREES

A <u>tree</u> is a connected graph with no cycles.



Properties:

a) Any two vertices are linked by a unique path.

b) Removing any edge disconnects the tree "minimally connected"

c) Adding any edge (x,y) creates a cycle.

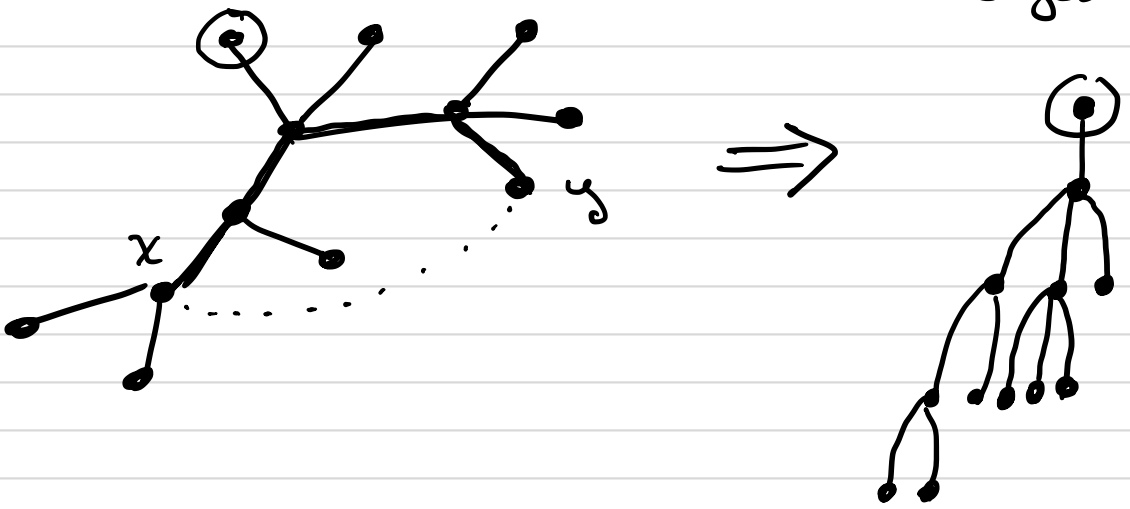
d) $|V| = |E| + 1$

Proof of (a):

1. Since G is connected there is at least one path between any pair.
2. Suppose there are two paths P_1 & P_2 between x & y.
3. The combined path forms a cycle
4. But a tree has no cycle, so (2) cant be true.

If G is a Tree, then $|V| = |E| + 1$

"Proof" by thought experiment: Imagine vertices are beads and edges are strings.



Every string (edge) has a bead (vertex) hanging at the bottom plus there is the one bead you are holding.

Proof by Induction

We want to prove that a bunch of related statements. $S_1, S_2, S_3, \ldots$ are all true.

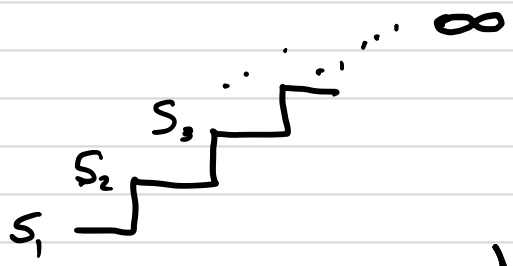
Proof: (Induction)

1) Prove S_1

2) For $k \geq 1$, Prove $S_{k-1} \Rightarrow S_k$ (OR: $S_k \Rightarrow S_{k+1}$)

It follows that S_n is true for all $n \geq 1$

Proof by induction is like climbing an infinite staircase.



Each step depends on the step below. A variation is known as Proof by <u>Strong</u> Induction.

Proof (Strong Induction)
1) Prove S_1 or the 1st several S_n, e.g. S_1, S_2, S_3
2) For any $k \geq 1$, Prove S_1 and S_2 and ... $S_{k-1} \Rightarrow S_k$

Revisiting Trees: $|V| = |E| + 1$ $\left(|E| = |V| - 1\right)$

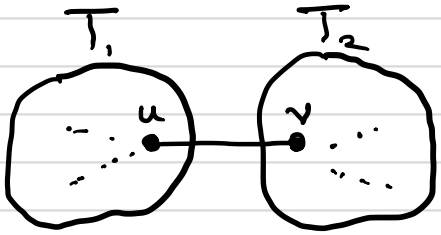
S_n : A tree with n vertices has $n-1$ edges.

Proof by Strong Induction.

S_1 :  (True)

S_2 :  (True)

S_3  (True)

S_k: 

$1 \leq k_1 < k$ $\quad$ $1 \leq k_2 < k$

since $k_1 + k_2 = k$

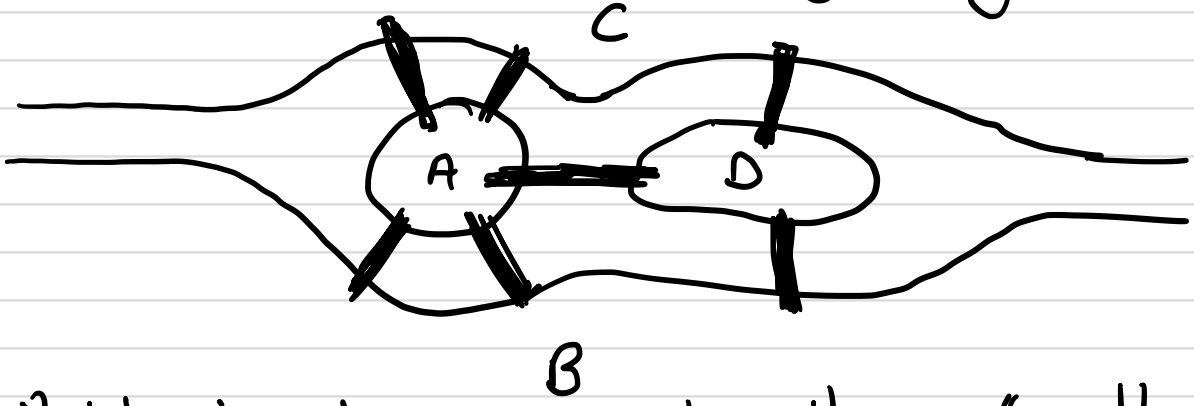
$$\underbrace{(k_1 - 1)}_{\#\text{ edges in } T_1} + \underbrace{(k_2 - 1)}_{\#\text{ edges in } T_2} + \underbrace{1}_{\text{bridge}} = k_1 + k_2 - 1 = k - 1$$

$\therefore S_k$ is true.

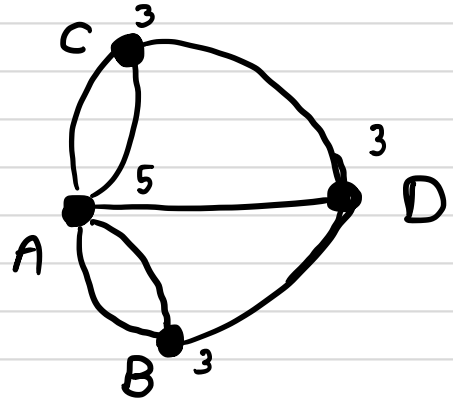
$S_1 \& S_2 \& S_3 \Rightarrow S_4$, $S_1 \& S_2 \& S_3 \& S_4 \Rightarrow S_5$, ... etc.

We conclude our study of Graph Theory by revisiting the problem that first inspired Euler.

The 7 Bridges of Königsberg (7BK)



Which is abstracted to the "multigraph":



Proof by Contradiction

To prove P, assume ¬P.

⋮

Derive: Q and ¬Q

Prove: 7Bridges of Königsburg has no Euler Path

Proof By Contradiction:

Suppose 7BK has an Euler Path.

a) Every Euler path begins and ends on some vertex, maybe the same vertex if its a circuit)

b) In the 7BK problem, there are 4 odd-degree vertices

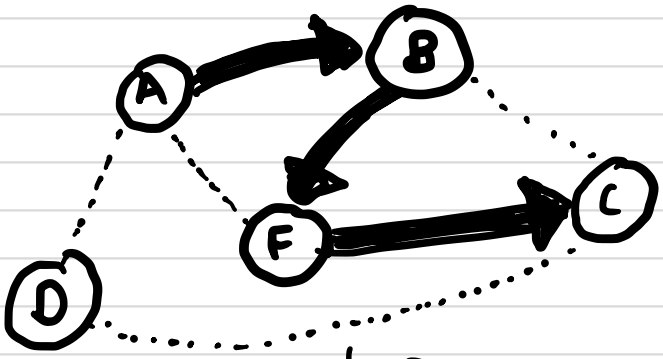
c) ∴ 2+ vertices are <u>neither the beginning or end</u> of path
Pick one, call it v

d) The path through v eliminates 2 edges with each visit, eventually leaving 1 edge.

e) On next visit to v, Path ends (no exit)
which contradicts our claim that v is not begin or end.
∴ our supposition is impossible! so its opposite ⇒ True ∎

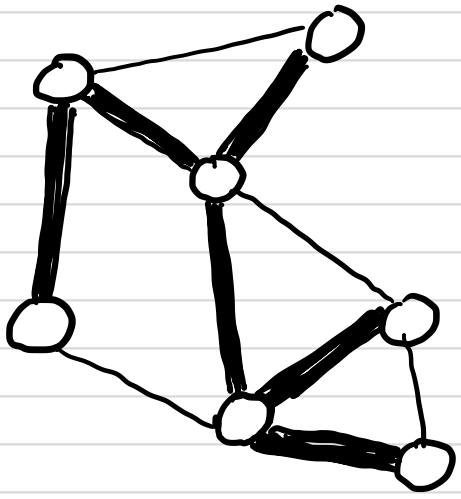
(A) Why learn Graph Algorithms?*

Just as graph theory is a rich area for mathematics research, graph algorithms are an essential aspect of computer science with many important applications:



A path from A⇝C

- Finding the shortest path from $X \overset{P_{shortest}}{\rightsquigarrow} Y$
- Communications protocols for distributed networks
- Modelling of a biological process - identifying drug targets.
- Analyzing social networks and how ideas or diseases spread.
- Planning robotic movements
- Defining topologies for computer networks, electrical grids, subways etc. that efficiently connects or "spans" the different sites we want to be able to visit

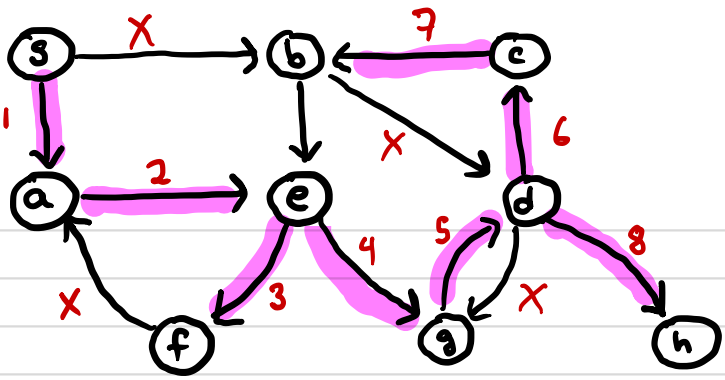


A "spanning" tree

---

* Note: Today is just an intro. You'll learn more when you take an algorithms class.

Graph Traversal: Visit All vertices



Method 1: Depth-First Search

$O(V+E)$

We use a stack (First-In Last-Out) to track our progress

Stack:

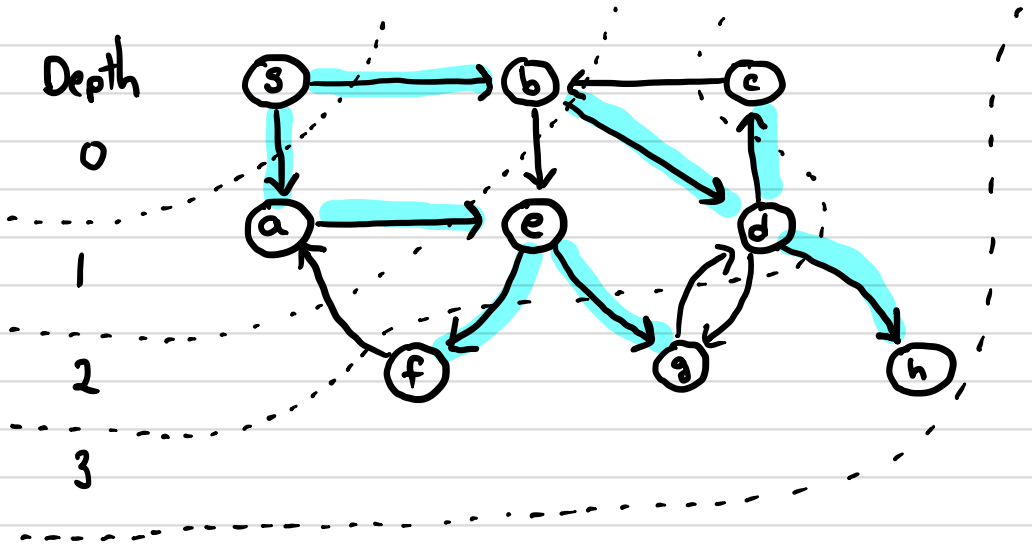
s	push s
s a	push a
s a e	push e
s a e f	push f
s a e	pop f
s a e g	push g
s a e g d	push d
s a e g d c	push c
s a e g d c b	push b
s a e g d c	pop b
s a e g d	pop c
s a e g h	push h
s a e g	pop h
s a e	pop g
s a	pop e
s	pop a
–	pop s (Done)

Notes:

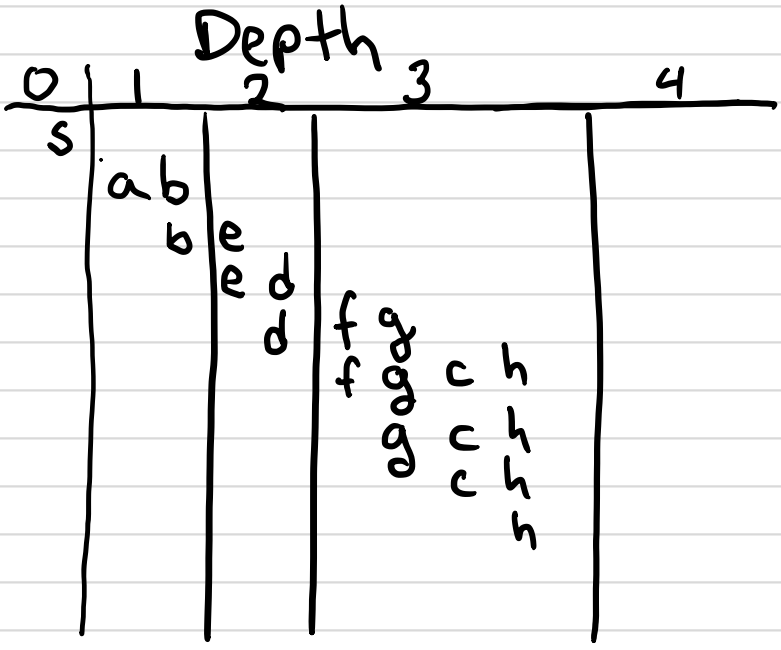
- This is an aggressive search. We go as far as we can until we can go no further, backtracking to the previous vertex.
- All vertices visited, not all edges.
- An adjacency list representation is very convenient. Why?

Graph Traversal : Visit all vertices

Method 2: Breadth-First Search $O(V+E)$



This time we use a <u>queue</u> (First In, First Out) to track our progress.



dequeue	enqueue
-	s
s	a, b
a	e
b	d
e	f, g
d	c h
f	-
g	-
c	-
h	-

Notes:
- BFS is "timid": search fully to depth k before moving on to depth k+1.
- We get the shortest path from s to every vertex as a side bonus!
- Searching a flight with no more than 1 stop (max depth = 2) would use BFS.

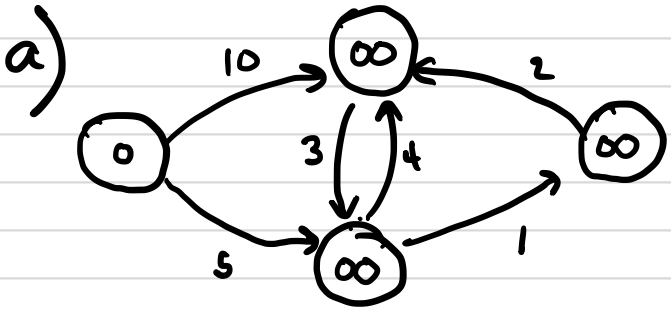
Shortest Path (Weighted Edges)

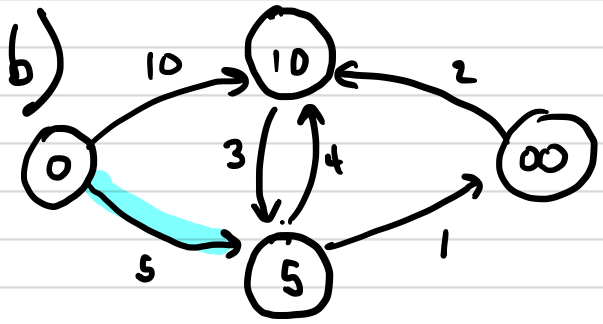
BFS solves the shortest path problem when the edges are unweighted (or weights are all '1.'

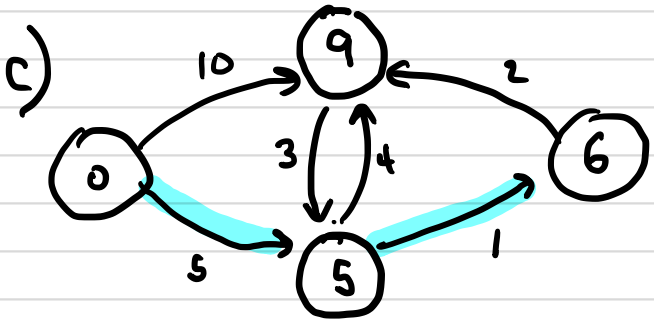
The famous Dijkstra's Shortest Path Algorithm solves the problem for edge weights $w \geq 0$.

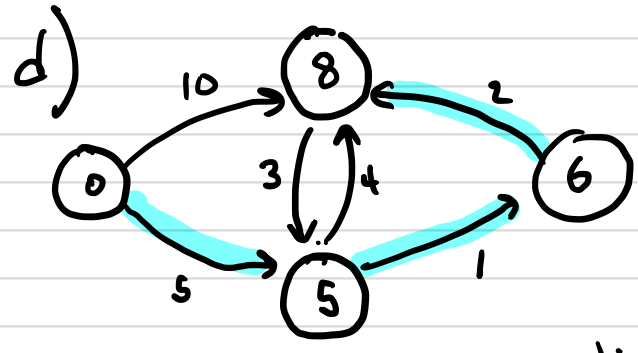
Running time is $O(|V| \cdot |E|)$ for standard implementations, but we can do better with fancy data structures, e.g. $O(|E| + |V| \log |V|)$

A sketch of how it works:









Idea: Work on the vertex closest to start, updating distances to adjacent nodes as we go along.