Here is a detailed and topographically accurate map of the city of Konigsberg (now called Kaliningrad) as it was in the year 1736. As you can see, the river Pregel runs through the middle of the city, and two islands sit in the middle of the river. The islands are connected to each other and to the banks of the river by seven bridges.

The citizens of Konigsberg were consumed by an apparently unsolvable problem: was it possible to leave one’s home, cross each bridge exactly once and return home again? Apparently many people claimed to have done it but nobody could quite remember how.

In 1736 Leonhard Euler put an end to the raging controversy by proving that it was not possible – in fact it was not even possible to go for a walk that started anywhere and ended anywhere and crossed each bridge exactly once. In the process of proving this, Euler invented graph theory.
Here’s how he did it. He simplified the map to look like this:

The circles (which could be squares, triangles, dots, whatever) are called **vertices** (the singular form is **vertex**) and the lines are called **edges**. Vertices are sometimes called **points**.

Two vertices that are joined by an edge are said to be **adjacent**. They are also called **neighbours**.

Here is a proof of Euler’s result.

Suppose it were possible to start at any vertex, follow an edge to another vertex, then follow another edge to another vertex, and so on until all edges had been used exactly once, and end up back where we started. Consider the **second** vertex we visit: every time we “arrive” at this vertex we have to leave again. This means that this vertex must have an even number of edges touching it. But in the diagram (and also on the map) we can see that every vertex has an odd number of edges touching it. This means no vertex can be the second one in the sequence ... which is not possible.

Now for some formal definitions:

A **graph** consists of a set of vertices V and a set of edges E, where E is a set containing two-element subsets of V. Sometimes we describe edges as **unordered pairs** of vertices.

We often think of a graph as a drawing, but it is important to remember that the graph is defined by the sets V and E, not by a particular drawing of them. A graph can be drawn in many different orientations and sizes, but it is still the same graph.

For example the graph G=(V,E) where V = {a,b,c,d} and E = { {a,b}, {a,c}, {b,c}, {c,d} } can be drawn many ways, including these:
This definition of “graph” is the one used in our textbook. It places a couple of implicit restrictions on the set of edges:

- all edges must be distinct – because E is a set, no edge can be duplicated
- no edge can join a vertex to itself – because each element of E is a two-element subset of V
- each edge is unordered – so an edge joining vertex x to vertex y can be written either as \{x,y\} or as \{y,x\} .... note that sometimes edges are written using simple parentheses, such as (x,y) or (a,b)

Note that by this definition, Euler’s diagram – the first graph ever – is not a graph because it has multiple edges with the same end-vertices.

Most mathematicians when writing about graph theory define graphs in a way that allows multiple edges between vertices (often called parallel edges) and edges that join vertices to themselves (often called loops) ... but then go on to say “but we will only consider graphs without parallel edges and loops”. In other words they define the more general class, but then restrict themselves to graphs as Scheinerman defines them. Our text just jumps directly to the definition that is most commonly used in practice.

For the record, most mathematicians would say that Scheinerman is defining simple graphs.

There are several more definitions and bits of notation to get out of the way.

The number of vertices in a graph G is sometimes called the order of G, and sometimes written as \(v(G)\), and sometimes written as just the letter \(n\). \(|V|\) is also used.

The number of edges in a graph G is sometimes called the size of G, and sometimes written as \(e(G)\), and sometimes written as just the letter \(m\). \(|E|\) is also used.
A walk in a graph is a sequence of edges such that the first edge shares a vertex with the second edge, the second edge shares its other vertex with the third edge, and so on. In a walk, vertices and edges may be used multiple times. Our text gives a different but equivalent definition: a walk is a sequence of vertices in which each vertex is adjacent to the previous vertex in the sequence.

A path is a walk that uses each edge and vertex at most once.

A circuit is a walk that returns to its starting vertex.

A cycle is a path that returns to its starting vertex (thus bending the path rule a little bit). Another way to define a cycle is “a path with end-vertices x and y, plus the edge {x,y}”

The degree of a vertex is the number of neighbours it has. We usually use \( d(v) \) to represent the degree of vertex \( v \).

Two vertices are connected if there is a path that starts at one and ends at the other.

**Theorem:** Let \( G \) be a graph. Then \( \sum_{v \in V} d(v) = 2 \cdot e(G) \)

**Proof:** The left side is the sum of the degrees of the vertices of \( G \). Every edge gets counted twice in this sum (for example, an edge \( \{x,y\} \) contributes 1 to the degree of \( x \) and 1 to the degree of \( y \)). Thus the sum of the degrees is twice the number of edges.

This has an interesting corollary:

**Corollary:** Let \( G \) be a graph. Then the number of vertices of \( G \) with odd degree is even.

**Proof:** Suppose \( G \) has an odd number of odd-degree vertices. Then \( \sum_{v \in V} d(v) \) must be odd, which contradicts the theorem we just proved. Therefore \( G \) cannot have an odd number of odd-degree vertices.
Euler Tours and Hamilton Cycles

Let’s revisit the problem that Euler solved for the city of Konigsberg, but now we can apply it to all graphs:

Given a graph G, does G contain a circuit that contains every edge exactly once?

Such a circuit is called an Euler Tour. A graph that has an Euler Tour is said to be Eulerian.

This is a very practical problem. For example, the graph might represent the streets of a city, and we might need to plan a route for the Recycling Service – they need to drive along every street, but it’s a waste of time to drive along any street more than once. Ideally they want a route that lets them drive along every street exactly once. Google Street View vehicles and postal delivery services have the same goal. A company called HiBot is developing robots that will crawl through water-supply pipes to determine their integrity (https://www.constructiondive.com/news/this-amphibious-robot-can-crawl-through-pipes-to-collect-data/438168/) - planning a route that explores every pipe in a network with minimal repetition will be an important task.

Before discussing this problem, let’s look at another, similar-looking problem:

Given a graph G, does G contain a cycle that contains every vertex? (Remember, the definition of a cycle precludes the possibility of including any vertex more than once.)

Such a cycle is called a Hamilton Cycle, and a graph with a Hamilton Cycle is said to be Hamiltonian. Hamilton was a famous mathematician and one of the pioneers of graph theory. He once marketed a board game based on finding Hamilton Cycles in graphs – for some reason it was not a million-seller.

This is also a very practical problem. If a delivery service needs to visit the same set of locations every day, the ideal route would be one that never needs to “double back” and visit some location twice on the way to other locations.

The reason for introducing both of these questions at the same time is to think about the relative difficulty of answering them. It may seem that the Hamilton Cycle problem is easier because it only involves finding a permutation of the vertices, whereas the Euler Tour problem involves ordering all the vertices and all the edges – and since it is a circuit, each vertex may occur multiple times in the solution.
In fact, exactly the opposite is true: for an arbitrary graph G, the Euler Tour question can be answered extremely easily, whereas the Hamilton Cycle question is so difficult that most mathematicians believe that there will never be an efficient method for answering it.

This is one of the things that I find so fascinating about graph theory: it is full of counter-intuitive results.

You will have to wait until CISC-365 for evidence that the Hamilton Cycle problem is so difficult, but the easiness of the Euler Tour question is well within our scope.

Theorem: Let G be a connected graph. G is Eulerian if and only if every vertex has even degree.

Proof:

First, observe that if G is Eulerian, then every vertex must have even degree, since the tour must pair up the edges at each vertex into “arrive/depart” pairs.

Now we show that if G is connected and all degrees are even, then G is Eulerian.

We will use a minimal counter-example proof, based on the number of edges in the graph. For a base case, we will consider graphs with 3 edges. A bit of experimentation shows that there are very few connected graphs with 3 edges, and only one of them has all degrees even: $K_3$. This is also the only one that is Eulerian. So the claim is true for $m = 3$.

Suppose G is a counter-example, and let G have the smallest number of edges in any counter-example. As always, we will use m to represent the number of edges in G.

We can show G contains a cycle: we can just start following edges – eventually we will come back to some vertex we have visited before, because every time we “arrive” at a vertex we can leave on a different edge (because the degree is even) and the graph is finite ... so we can’t keep visiting new vertices forever.

Let C be a cycle in G. Let G’ be the graph that results in removing the edges of C. This reduces the degree of each vertex in C by exactly 2 – so G’ is a graph in which each vertex has even degree, and G’ has < m edges ... so G’ is not a counterexample.

Suppose G’ is connected. Then G’ has an Eulerian Tour. We can extend that to an Eulerian tour of G by following the Tour of G’ until we reach one of the vertices in C ... then follow the edges of C back to that vertex ... then follow the rest of the Tour of G’. This completed tour includes every edge of G exactly once, so G is Eulerian (and therefore not a counter-example).
Now suppose \( G' \) is not connected. Each of its connected components is too small to be a counterexample, so each one has an Eulerian Tour. I leave it as an exercise for you to determine how we can use these Eulerian Tours of the components of \( G' \), plus the cycle \( C \), to construct an Eulerian Tour of \( G \). (Again, this means \( G \) is not a counter-example.)

Thus we find that the minimal counter-example is not a counter-example ... and this contradiction allows us to conclude that no counter-examples exist.

**Graph Isomorphism**

Suppose we are given two graphs, both on the same number of vertices with the same number of edges. Sometimes it is extremely useful to know if they are actually the same graph – for example, if the graphs represent molecular structures, it is useful to know if they are actually the same compound. If they are then any property know to be held by one (for example, specific gravity) will be shared by the other.

This figure, repeated from above, shows three graphs that are all the same graph: they have the same vertices and the same edges. But suppose we threw away the labels, or suppose we used different labels on one of the drawings. Would we still be able to say the graphs were related?

We say that two graphs are **isomorphic** if we can label the vertices in the two graphs with the same labels in such a way that all the adjacencies are preserved: each pair of vertices are adjacent in one graph if and only if they are adjacent in the other graph. The notation for isomorphism is \( G \cong H \).
As an exercise, consider these three graphs. Each has six vertices and eight edges. Are any of them isomorphic to each other?

Graph isomorphism is not only a practical problem – it is also of enormous theoretical interest. We have seen questions that are easy to answer (such as “Is G Eulerian?”) and questions that are extremely difficult to answer (such as “Is G Hamiltonian?”). The question “Are $G_1$ and $G_2$ isomorphic?” is unusual in that it seems to float somewhere between these two extremes ... but nobody knows for sure. It is entirely possible that somebody will discover an easy way to answer this question, but it is also possible that somebody will prove that there is no easy solution. Here again we find ourselves asking simple questions that reveal just how much undiscovered country there is in the field of discrete mathematics.
Subgraphs

As you might guess, a subgraph is part of a graph (the relationship between graphs and their subgraphs is almost identical to the relationship between sets and their subsets).

Formally, let $G = (V,E)$ and $H = (X,F)$ be graphs. If $X \subseteq V$ and $F \subseteq E$ then $H$ is a subgraph of $G$

In plain words, this means $H$ is a subgraph of $G$ if every vertex and edge of $H$ is also in $G$.

A subgraph $H$ of $G$ can contain some, all, or none of the edges of $G$ that join the vertices in $H$.

For example, consider this graph $G$. Each of the graphs $H_1$, $H_2$ and $H_3$ is a subgraph of $G$ – and there are many many more subgraphs not shown here. As a practice problem, compute the total number of subgraphs of this graph.

![Graph G with subgraphs H1, H2, and H3]

Note that a subgraph is not allowed to “invent” edges that were not in $G$, so for the graph shown above, no subgraph can contain an edge joining (for example) $a$ and $d$.

There is something fundamentally different about the subgraph $H_2$ in the diagram, that distinguishes it from $H_1$ and $H_3$. In both $H_1$ and $H_3$, potential edges that could have been in the subgraph have not been included. For example, the edge $\{a,c\}$ is in $G$ and both these vertices are in $H_1$, so this edge could have been included when $H_1$ was defined. Similarly,
given that H3 contains the vertices b and c, the edge {b,c} could have been included when H3 was defined.

But the edge set of H2 contains every allowable edge between the vertices a, b and d (remember we are only allowed to use edges that are in G).

A subgraph that contains every allowable edge between its vertices is called an induced subgraph. An induced subgraph is completely determined by its vertex set – once we have decided which vertices are in the subgraph H, the edges of H are exactly the edges of G that join the chosen vertices.

Formally, if H = (Z,F) is a subgraph of G = (V,E) and F = (Z × Z) ∩ E, then H is an induced subgraph of G.

Question: If G is a graph with n vertices, how many induced subgraphs does G have?

One more definition regarding subgraphs: Let G be a graph and let H be a subgraph of G. If the vertex set of H is exactly the vertex set of G, then H is called a spanning subgraph of G.

Spanning subgraphs are particularly important when we are using graphs to represent communication or transportation networks. If each vertex represents a city on a map and the edges represent highways, we often need to explore subgraphs that include all the cities but possibly leave out some of the highways ... these are spanning subgraphs!

Question: If G is a graph with n vertices, how many induced spanning subgraphs does G have?

Note that
- every graph is a subgraph of itself (so the “subgraph of” relation is reflexive),
- if H is a subgraph of G, and G is a subgraph of F, then H is a subgraph of F (so the “subgraph of” relation is transitive)
- if H is a subgraph of G and G is a subgraph of H, then G and H are the same graph (so the “subgraph of” relation is anti-symmetric)

So the “subgraph of” relation is a partial order!
Cliques, Independent Sets and Complements

We call a graph **complete** if it contains all possible edges. The notation for the complete graph on \( n \) vertices is \( K_n \).

If \( H \) is a subgraph of \( G \) and \( H \) is a complete graph, we say \( H \) is a **\( k \)-clique** of \( G \). (Some people use the term \( k \)-clique as a synonym for \( K_n \) --- that’s ok.)

Side note: an innocent-sounding question is “Given a graph \( G \), what is the largest \( k \)-clique that is a subgraph of \( G \)?” Simple as it sounds, this question is so difficult that nobody has ever discovered a good way to find the answer, despite decades of research. This problem actually falls into the same category of problems as “Given a partial order \( P \), what is the dimension of \( P \)?”: they are both among the hardest problems in computer science.

Why is this problem even interesting? It turns out that it is relevant to lots of very practical problems. For example, suppose the vertices of the graph represent Queen’s courses, and two vertices are joined by an edge if and only if there is at least one student taking both courses. When it comes time to schedule final exams, any vertices that are adjacent must be scheduled into different time-slots. So if there is a \( k \)-clique, the exam schedule will need at least \( k \) time-slots. Knowing the size of the largest \( k \)-clique would give a lower bound on the number of time-slots needed.

A subgraph with \( k \) vertices and **no edges** is called a **\( k \)-independent set**. \( K \)-independent sets are also useful in applications such as the exam-scheduling problem we just looked at. If we
can find the largest independent set in \( G \), those vertices represent a set of courses that can all be scheduled into the same exam slot.

Unfortunately, finding the largest independent set in a graph \( G \) is \textbf{just as hard} as finding the largest \( k \)-clique in \( G \).

**Complement**

Let \( G = (V, E) \) be a graph. We define the complement of \( G \) as \( \bar{G} = (V, \bar{E}) \) where \( \bar{E} \) contains all the edges that are not in \( E \). As an exercise, determine the complement of some of the graphs in these notes.

It should be easy to see that the complement of a \( k \)-clique is a \( k \)-independent set and vice versa, and that \( H \) is a \( k \)-clique in \( G \) if and only if \( \bar{H} \) is a \( k \)-independent set in \( \bar{G} \).

This means that if we could find the largest \( k \)-clique in all graphs, we could use this to find the largest independent set as follows:

- to find the largest independent set in \( G \),
  - construct \( \bar{G} \)
  - find the largest \( k \)-clique in \( \bar{G} \)
  - these vertices form the largest independent set in \( G \)

You should convince yourself that this works in reverse also: if we could find the largest independent set in all graphs, we could use this to find the largest clique in another graph.

This idea – two different-seeming problems that turn out to be closely related – will be crucial in CISC-365.
Connectivity

... more definitions ...

Recall that two vertices \( x \) and \( y \) are **connected** if there is a path from \( x \) to \( y \).

Note that a vertex \( x \) is always connected to itself (by a path of length 0)

Note that if \( x \) is connected to \( y \), then \( y \) is connected to \( x \)

Note that if \( x \) is connected to \( y \) and \( y \) is connected to \( z \), then \( x \) is connected to \( z \).

So the “connected to” relationship is reflexive, symmetric and transitive ...i.e. it is an equivalence relation.

The equivalence classes of the “connected to” relation define what we call the **connected components** of the graph \( G \).

More intuitively, each connected component of \( G \) is a maximal set of vertices that are all connected to each other.

We say that \( G \) is **connected** if \( G \) has exactly one connected component (i.e. for every pair of vertices \( x \) and \( y \), \( G \) contains an \( xy \)-path).

Let \( G = (V,E) \) be a graph. If \( x \) is a vertex of \( G \) such that if we delete \( x \) from \( V \) (which means we also have to delete all edges that touch \( x \)) the number of connected components increases, then we call \( x \) a **cut-vertex** of \( G \). Similarly, if \( e \) is an edge such that deleting \( e \) from \( E \) increases the number of connected components, we call \( e \) a **cut-edge** or **bridge** of \( G \).
In the graph shown above, the cut-vertices are x, y and z. The only cut-edge is e.

Cut-vertices and cut-edges are important because in a communication or transportation network, they are the bottlenecks.

**Trees**

Recall that a cycle is a path that begins and ends at the same vertex (and contains no vertex more than once).

A connected graph that has no cycles is called a **tree**.

A graph that has no cycles is called a **forest**. Each connected component of a forest is a tree.
There are a number of different ways to define trees:

A tree is a graph in which every pair of vertices is joined by exactly one path.

A tree is a connected graph in which every edge is a cut-edge.

A tree is a connected graph in which \(|E| = |V| - 1\)

We can prove that each of these is equivalent to the original definition (“a tree is a connected graph without cycles”)

Leaves

A vertex of degree 1 in a tree is called a **leaf** of the tree.

**Theorem:** Let \( T \) be a tree with \( \geq 2 \) vertices. Then \( G \) contains at least two leaves.

**Proof:** Let \( P \) be the longest path in \( T \), and let vertices \( x \) and \( y \) be the first and last vertices in \( P \).

Suppose \( x \) is not a leaf. Then \( x \) has degree \( \geq 2 \), which means that \( x \) has a neighbour that is not in \( P \) (if all neighbours of \( x \) are in \( P \), then the graph contains a cycle). Let \( z \) be this neighbour of \( x \). Then since \( z \) is not in \( P \), we can create a new path \( P' \) by adding the edge \( \{xz\} \) to \( P \). But \( P' \) is longer than \( P \), which is impossible since we chose \( P \) to be the longest path in \( T \). Therefore \( x \) does **not** have any neighbour other than the one in \( P \). Therefore \( x \) has degree 1, so it is a leaf. Now we can repeat the same argument for vertex \( y \), so \( y \) is also a leaf.

In future courses we will spend a lot of time looking at **spanning trees of a graph** ... which are defined to be spanning subgraphs that are also trees.
Here’s an important result about trees:

**Theorem:** Let $T$ be a tree on $\geq 2$ vertices, with leaf vertex $x$. Then $T - x$ is a tree.

**Proof:** We need to show that $T - x$ is connected and has no cycles. To show that $T - x$ is connected, let $a$ and $b$ be any two vertices in $T - x$. Since they are in $T - x$, they are also in $T$, which means $T$ contains an ab-path $P$. Since $x$ has degree 1, $x$ cannot be in $P$, so $P$ is completely in $T - x$. Thus all pairs of vertices in $T - x$ are connected, so $T - x$ is connected. Showing that $T - x$ has no cycles is trivial: $T$ has no cycles since it is a tree, and deleting $x$ cannot create a cycle ... so $T - x$ has no cycles. Thus $T - x$ is a connected graph with no cycles ... which means it is a tree.

This is important because it lets us use induction to prove many properties of trees. For example ...

**Theorem:** Let $T$ be a tree on $n$ vertices. Then $|E| = |T| - 1$

**Proof:** We will use induction on the number of vertices in the tree.

Base case: The only tree with one vertex is $K_1$ ... which has one vertex and zero edges, so the claim is true when the number of vertices in the tree is one.

Inductive Assumption: Assume the claim is true for all trees on $\leq k$ vertices.

Let $T$ be a tree on $k+1$ vertices. Since $k+1 \geq 2$ we know $T$ has a leaf $x$, and we know $T - x$ is a tree! And we know $T - x$ has exactly $k$ vertices!! So we know that the number of edges in $T - x$ is exactly $k - 1$ (by the Inductive Assumption)!!! So $T$ has $k+1$ vertices and $k$ edges!!!! So the claim is true for $T$ !!!!!

Therefore the claim is true for all trees !!!!!

And now, like our distant ancestors millions of years ago, it is time to leave the safety of the trees and venture out into the wide plains ... where we will need a map.
Here’s a problem that has been intriguing graph theorists for over 250 years. Suppose we have a map of a continent divided up into countries. We will assume that each country consists of a connected piece of land (unlike the USA and Russia, both of which have disconnected bits that are separate from the main territory). The map might look something like this:

Now suppose we are cartographers and we want to make a coloured map of this continent, but we want to make sure that two countries that share a border never have the same colour. The question is, how many colours do we need?
Here’s a colouring of that map that uses 4 colours ... but could we do it with 3? The answer is no ... can you see why?

Now what does this have to do with graph theory? Well we can represent the map with a graph by creating a vertex for each country, and making two vertices adjacent if the countries they represent share a border.
**Definition:** A proper colouring (or legal colouring) of a graph is an assignment of colours to vertices so that no adjacent vertices have the same colour.

We have seen that the graph above can be properly coloured using 4 colours, but it can also be properly coloured using 5 or 6 or 7 etc, colours. Since there are 13 countries, we could use 13 colours – giving each vertex an unique colour. We could even say it is possible to properly colour this graph with 50 colours, as long as we don’t mind having a bunch of colours left unused.

**Definition:** The chromatic number of a graph $G$ is the smallest integer $k$ such that $G$ can be properly coloured with $k$ colours. We write this as $\chi(G) = k$ (that symbol is the Greek letter “chi”, pronounced “kye”)

So for the graph $G$ shown above, $\chi(G) = 4$.

How large can $\chi(G)$ be? In the complete graph $K_n$ every vertex is adjacent to all other vertices so every vertex must have its own colour. Thus $\chi(K_n) = n$
How small can $\chi(G)$ be? In the complement of $K_n$, there are no edges at all so every vertex can have the same colour. Thus $\chi(K_n) = 1$

Now we can formalize the question we started with. Let $G$ be a graph corresponding to a map as we defined it above. What is the maximum possible chromatic number of $G$?

People started to ask this question in the 1800’s. It was easy to make maps like the one above that have chromatic number = 4, but nobody ever found one with chromatic number = 5. Thus a conjecture was born: No map needs more than 4 colours. (In our notation, if $G$ is a graph corresponding to a map, then $\chi(G) \leq 4$. ) This became known as the **4-colour conjecture**.

In 1852, a man named **Francis Guthrie** sent the question to the man who had taught him mathematics: **Augustus De Morgan** (the De Morgan who discovered De Morgan’s Laws). He sent the question to **Hamilton**, who was a famous graph theorist, and Hamilton sent it to **Cayley**, an extremely important mathematician. None of them could resolve the conjecture.

Then in 1879, a mathematician named **Kempe** published a proof of the conjecture, which then became the **4-colour theorem** ... except it wasn’t. Kempe’s proof had an error. The error was discovered in 1890 by **Heawood** – at which point the theorem became a conjecture again.

And there it stayed until 1976, when two mathematicians named **Appel** and **Haken** finally and completely proved that the conjecture is true: there is no map that requires more than 4 colours. (Except that their proof was not quite complete ... but they patched the gaps within a few months.) So the conjecture is a theorem again, and will remain so.

On a personal note, I remember when this happened (the Appel and Haken proof, not the Kempe one!) I was an undergraduate student at the time, but already fascinated by graph theory. It was very exciting to know that such a long-standing mystery had finally been solved. The post-office in the city where Appel and Haken did their work changed their stamp to read “4 Colors Suffice” (since they were in the USA, the word “colours” was spelled incorrectly).

The Appel and Haken proof was controversial because it was the first “computer proof”: their proof involved finding proper 4-colourings for a very large set of fairly large graphs – too many for a human to ever check by hand. Some critics rejected the proof on the grounds that it could not be verified by humans. However, soon other people wrote independent programs to repeat and verify the result, and now I don’t think there is anyone who disputes it. Nonetheless I suspect that a lot of mathematicians still long for a short and elegant proof of the theorem.
You can read a more detailed history of the conjecture and how it became a theorem here: http://www-history.mcs.st-and.ac.uk/HistTopics/The_four_colour_theorem.html

This is as far as we got in class on November 27, so this is the cut-off point for material that will be on Thursday’s test. I’m including notes on the rest of the material I had planned to cover, in case you are interested (I think this graph colouring material is fascinating, but maybe that’s just me ...

Anyway, back to our introduction to colouring. We have seen that \(\chi(G)\) can be any number from 1 to \(n\) (where \(n\) is the number of vertices in \(G\)).

We can put a better upper bound on \(\chi(G)\):

**Theorem:** Let \(\Delta\) be the maximum vertex degree in \(G\). Then \(\chi(G) \leq \Delta + 1\)

**Proof:** We will present an argument that it is possible to construct a proper \(\Delta + 1\) colouring of \(G\).

Pick any vertex and give it a colour. Now pick another vertex and give it any colour that is not already assigned to one of its neighbours. Continue to pick vertices and colour them. On each iteration of this process, the neighbours of the vertex can use at most \(\Delta\) different colours because the vertex has at most \(\Delta\) neighbours. Thus there must be a colour available for the vertex we are colouring.

So for example, if \(G\) has \(10^6\) vertices but \(\Delta = 8\) we know \(\chi(G) \leq 9\) without having to do any work at all.

It is natural to ask “Given \(G\), how can we compute \(\chi(G)\) exactly?” … and this is a great question.

We can easily answer the specific question “Given \(G\), does \(\chi(G) = 1?\)” … because \(\chi(G) = 1\) if and only if \(G\) has vertices but no edges (since any edge forces its ends to have different colours).

We can also answer the question “Given \(G\), does \(\chi(G) = 2?\)” … but it’s a bit harder.
We can determine if $\chi(G) = 2$ by attempting to build a proper 2-colouring of the graph, as follows:

Choose a vertex and colour it Red (the choice of vertex and colour doesn’t matter)

Now all neighbours of this vertex must be coloured Blue. If any of these vertices are adjacent to each other, they *can’t* both be Blue, but neither of them can be Red because they are adjacent to the vertex we started with. Thus if any of these vertices are adjacent to each other, $G$ cannot be properly 2-coloured.

If this test is passed, we look at all the neighbours of the vertices we just coloured Blue. All of these vertices *must* be coloured Red ... and if any of them are adjacent to each other, the colouring fails and $G$ cannot be properly 2-coloured.

We continue to colour sets of vertices in this manner. We either end with a proper 2-colouring of $G$, or we find two adjacent vertices to which we must assign the same colour, which tells us that $G$ does not have a proper 2-colouring.

The reason this gives us a definite answer about whether $G$ can be properly 2-coloured is because we never have any options or alternatives. Every time we reach a new vertex, we *must* assign it the colour that is different from the vertices we coloured on the previous iteration.

This is why this process doesn’t work if we want to answer “Given $G$, does $\chi(G) = 3$?” ... with 3 colours we have options.

We can start the same way: pick a vertex and colour it Red. But now when we look at the neighbours of this vertex, we don’t know whether we should colour them all Blue, all Green, or some combination of the two. If we guess, we may be wrong ... we may have to try out all possible combinations of Blue and Green on these vertices, and then for their neighbours, we might have to try all the combinations of Red, Blue and Green, and so on for each new group of vertices ... and trying all those combinations will take a long time. Even for relatively small graphs, this is infeasible.

The truth is that this question: “Given $G$, does $\chi(G) = 3$?” is recognized as one of the hardest problems in all of computing science. (I say “in computing science” instead of “in mathematics” because in computing we are particularly interested in finding algorithms to solve problems.)
Now this may seem a little bit familiar. Recall that when we looked at the dimension of posets, we found that it is easy to determine if a poset’s dimension is either 1 or 2, but we have no fast method to determine if a poset’s dimension is 3.

These two problems have the strange property that they are easy to solve for 1 and 2, but effectively impossible to solve for any value $\geq 3$. There are many other problems that have exactly the same property. Trying to figure out why this is – what causes this sudden and overwhelming increase in difficulty as we go from 2 to 3 – is an active field of research in computing science.

Coming back to Earth, we have one more bit of notation:

**Definition:** If a graph $G$ is 2-colourable, we call $G$ **bipartite**.

We often draw bipartite graphs like this, with all the Red vertices on one side and all the Blue vertices on the other. Drawn in this fashion, all the edges of the graph go between the two sets of vertices.
And one final question: All trees on \( \geq 2 \) vertices have the same chromatic number. What is it?