# Contents

# Chapter 1

# Introduction

The first chapter is an introduction, including the formal definition of a graph and many terms we will use throughout. More importantly, however, are examples of these concepts and how you should think about them. As a first nontrivial use of graph theory, we explain how to solve the "Instant Insanity" puzzle.

## 1.1 A first look at graphs

### 1.1.1 The idea of a graph

First and foremost, you should think of a graph as a certain type of picture, containing dots and lines connecting those dots, like so:

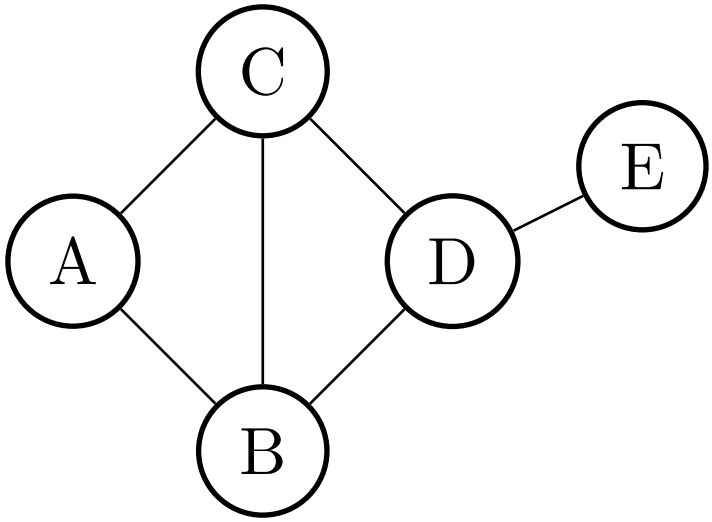



**Figure 1.1.1** A graph

We will typically use the letters $G, H$, or $\Gamma$ (capital Gamma) to denote a graph. The "dots" or the graph are called *vertices* or *nodes*, and the lines between the dots are called *edges*. Graphs occur frequently in the "real world", and typically how to show how something is connected, with the vertices representing the things and the edges showing connections.

- *Transit networks:* The London tube map is a graph, with the vertices representing the stations, and an edge between two stations if the tube

goes directly between them. More generally, rail maps in general are graphs, with vertices stations and edges representing line, and road maps as well, with vertices being cities, and edges being roads.

- *Social networks:* The typical example would be Facebook, with the vertices being people, and edge between two people if they are friends on Facebook.
- *Molecules in Chemistry:* In organic chemistry, molecules are made up of different atoms, and are often represented as a graph, with the atoms being vertices, and edges representing covalent bonds between the vertices.

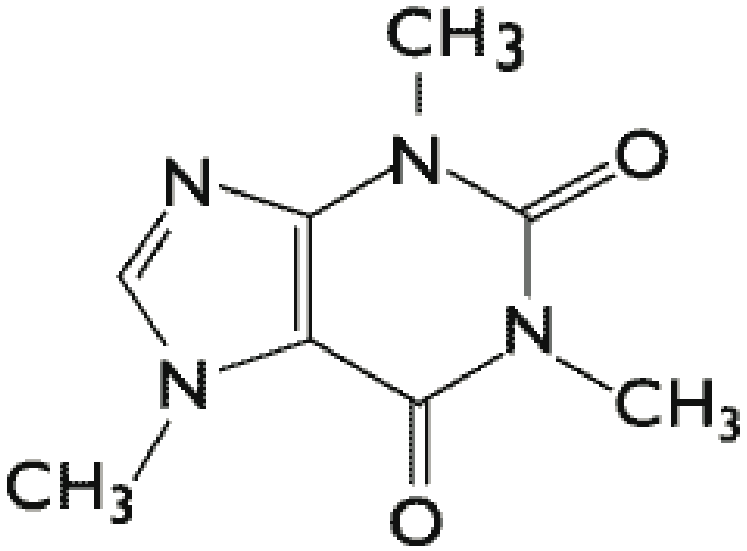



**Figure 1.1.2** A Caffeine Molecule, courtesey Wikimedia Commons

That is all rather informal, though, and to do mathematics we need very precise, formal definitions. We now provide that.

### 1.1.2 The formal definition of a graph

The formal definition of a graph that we will use is the following:

**Definition 1.1.3** A *graph* $G$ consists of a set $V(G)$, called the *vertices* of $G$, and a set $E(G)$, called the *edges* of $G$, of the two element subsets of $V(G)$ $\Diamond$

**Example 1.1.4** Consider the water molecule, which consists of a single oxygen atom, connected to two hydrogen atoms. It has three vertices, and so $V(G) = \{O, H1, H2\}$, and two edges $E(G) = \big\{\{O, H1\}, \{O, H2\}\big\}$ $\square$

This formal definition has some perhaps unintended consequences about what a graph is. Because we have identified edges with the two things they connect, and have a set of edges, we can't have more than one edge between any two vertices. In many real world examples, this is not the case: for example, on the London Tube, the Circle, District and Picadilly lines all connect Gloucester Road with South Kensington, and so there should be multiple edges between those two vertices on the graph. As another example, in organic chemistry, there are often "double bonds", instead of just one.

Another consequence is that we require each edge to be a two element subset of $V(G)$, and so we do not allow for the possibility of an edge between a vertex

and itself, often called a *loop*.

Graphs without multiple edges or loops are sometimes called *simple graphs*. We will sometimes deal with graphs with multiple edges or loops, and will try to be explicit when we allow this. Our default assumption is that our graphs are simple.

Another consequence of the definition is that edges are symmetric, and work equally well in both directions. This is not always the case: in road systems, there are often one-way streets. If we were to model Twitter or Instragram as a graph, rather than the symmetric notion of friends we would have to work with "following". To capture these, we have the notion of a *directed graph*, where rather than just lines, we think of the edges as arrows, pointing from one vertex (the source) to another vertex (the target). To model Twitter or Instagram, we would have an ege from vertex $a$ to vertex $b$ if $a$ followed $b$.

### 1.1.3 Basic examples and concepts

Several simple graphs that are frequently referenced have specific names.

**Definition 1.1.5** The complete graph $K_n$ is the graph on $n$ vertices, with an edge between any two distinct vertices. ◊

**Definition 1.1.6** The empty graph $E_n$ is the graph on $n$ vertices, with no edges. ◊

**Definition 1.1.7** The path graph $P_n$ is the graph on $n$ vertices $\{v_1, \ldots, v_n\}$ with edges $\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}\}$. ◊

**Definition 1.1.8** The cycle graph $C_n$ is the graph on $n$ vertices $\{v_1, \ldots, v_n\}$ with edges $\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}\}$. ◊

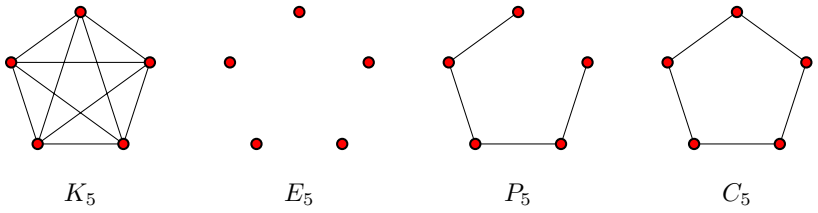
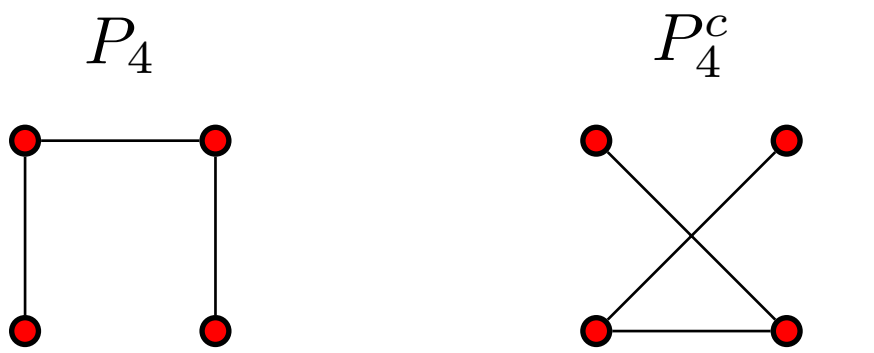



**Figure 1.1.9** Basic graphs

**Definition 1.1.10** The complelement of a simple graph $G$, which we will denote $G^c$, and is sometimes written $\overline{G}$, is the graph with the same vertex set as $G$, but $\{v, w\} \in E(G^c)$ if and only if $\{v, w\} \notin E(G)$; that is, there is an edge between $v$ and $w$ in $G^c$ if and only if there is not an edge between $v$ and $w$ in $G$ ◊

**Example 1.1.11** The empty graph and complete graph are complements of each other; $K_n^c = E_n$

The path graph $P_4$ and its complement are shown below:

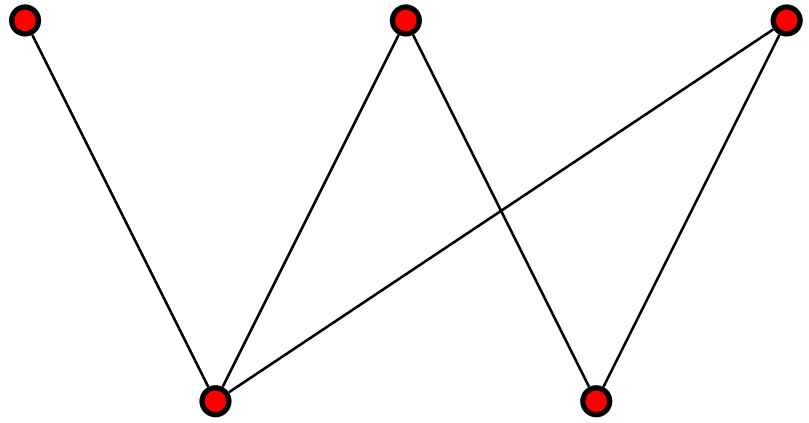



□

It commonly occurs that there are two different types of vertices, and the edges only go between vertices of the two types. For example, we may be modelling a company, and one type of vertices may represent the employees, and another type of vertices could represent the different jobs that need done, with an edge between a worker and a job if the worker is qualified to do that particular job. We call these graphs *bipartite*.

**Definition 1.1.12** A graph **G** is *bipartite* if its vertices can be coloured red and blue so that every edge goes between a red vertex and a blue vertex. ◊

**Example 1.1.13** The graph below is bipartite.



□

As another example, note that the cycle graph $C_4$ is bipartite -- we can colour vertices 1 and 3 red, and vertices 2 and 4 blue. But the cycle graph $C_3$ is *not* bipartite: as the two colours are interchangable, we can assume we coloured vertex 1 red; then since it is adjacent to both 2 and 3, those vertices must both be blue, but they're adjacent to each other, which violates the definition of bipartite. More generally, we have:

**Lemma 1.1.14** *The cycle graph $C_n$ is bipartite if and only if $n$ is even.*

*Proof.* Let's try to colour the vertices of $C_n$ red and blue so that adjacent vertices have different colour. Without loss of generality, we may assume that $v_1$ is coloured blue. Then since it is adjacent to $v_1, v_2$ must be coloured red. Continuing in this way, we see that $v_k$ is blue for odd $k$ and red for even $k$. But $v_n$ is adjacent to $v_1$, and so these will have different colours precisely when $n$ is even. ■

**Lemma 1.1.15** *A graph* **G** *is bipartite if and only if* **G** *has no subgraphs that are isomorphic to $C_{2k+1}$*

*Proof.* First, note that if $\mathbf{G}_2$ is a subgraph of $\mathbf{G}_1$, and $\mathbf{G}_1$ is bipartite, then so is $\mathbf{G}_2$: colouring the vertices of $\mathbf{G}_1$ red and blue in particular colours the vertices of $\mathbf{G}_2$ as well. Hence, we see if that $\mathbf{G}$ has a subgraph isomorphic to $C_{2k_1}$, which isn't bipartite by the previous lemma, then $\mathbf{G}_1$ can't be bipartite, either.

In the other direction, we assume that $\mathbf{G}$ has no subgraphs isomorphic to $C_{2k+1}$; we need to prove that $\mathbf{G}$ is bipartite. Again, let's try to colour the vertices of $\mathbf{G}$ red and blue so that adjancent vertices have different colours. Choose a vertex $v$ of $\mathbf{G}$, without loss of generality we may assume that $v$ is coloured blue; then all vertices adjacent to $v$ -- i.e., those vertices at distance 1 from $v$ -- are coloured red. The vertices adjacent to those must be blue, and the ones adjacent to those must be red, alternating out. This suggests trying to colour all vertices at odd distance from $v$ red, and those vertices at even distance from $v$ blue. We will show that if this colouring has two vertices of the same colour that are adjacent, then $\mathbf{G}$ has an odd cycle.

Assume that $u$ and $w$ are two vertices coloured red that are adjacent. Since $u$ is red, the distance from $v$ to $u$ is odd -- there is a path $v = v_0 - v_1 - \cdots - v_{2k+1} = u$. Similarly, there is an odd length path from $v$ to $w$: $v = w_0 - w_1 - \cdots - w_{2\ell+1} = w$. Taking the union of these two subgraphs and the edge connecting $u$ and $w$, we get a closed walk consisting of $(2k+1) + (2\ell+1) + 1 = 2k + 2\ell + 3$ edges, which is odd. This walk may repeat some vertices and edges, but if it does we can split it into two smaller walks, one of which must have odd length, and eventually we must get a closed walk of odd length that doesn't repeat any vertices.

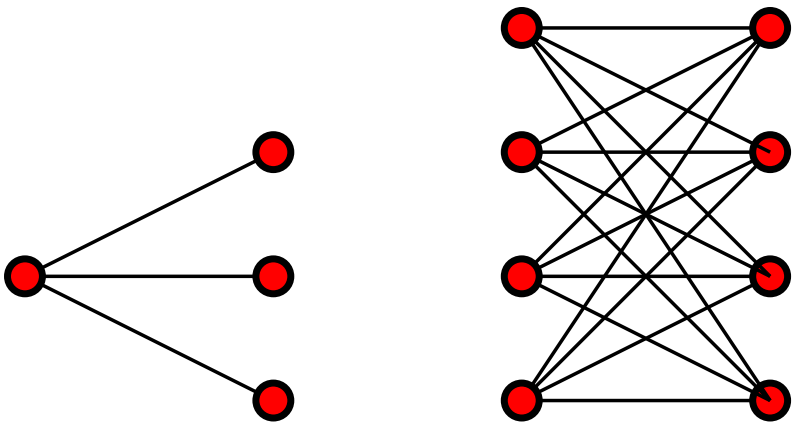
The case that $u$ and $w$ are both coloured blue is completely analogous, except we will be merging two paths with an even number of edges and one extra edge to get a path with odd length. ■

A special type of bipartite graph is the complete bipartite graphs $K_{m,n}$, which are the simple graphs that have as many edges as possible while still being bipartite.

**Definition 1.1.16** The *complete bipartite graph* $K_{m,n}$ is the graph with $m$ red vertices and $n$ blue vertices, and an edge between very red vertex and every blue vertex. ◊

**Example 1.1.17** The complete bipartite graph $K_{2,2}$ is isomorphic to $C_4$.

The graphs $K_{1,3}$ and $K_{4,4}$ are drawn below.



$K_{1,3}$ $K_{4,4}$

□

## 1.2 Degree and handshaking

### 1.2.1 Definition of degree

Intuitively, the *degree* of a vertex is the "number of edges coming out of it". If we think of a graph $G$ as a picture, then to find the degree of a vertex $v \in V(G)$ we draw a very small circle around $v$, the number of times the $G$ intersects that circle is the degree of $v$. Formally, we have:

**Definition 1.2.1** Let $G$ be a simple graph, and let $v \in V(G)$ be a vertex of $G$. Then the *degree of* $v$, written $d(v)$, is the number of edges $e \in E(G)$ with $v \in e$. Alternatively, $d(v)$ is the number of vertices $v$ is adjacent to. ◊

**Example 1.2.2**

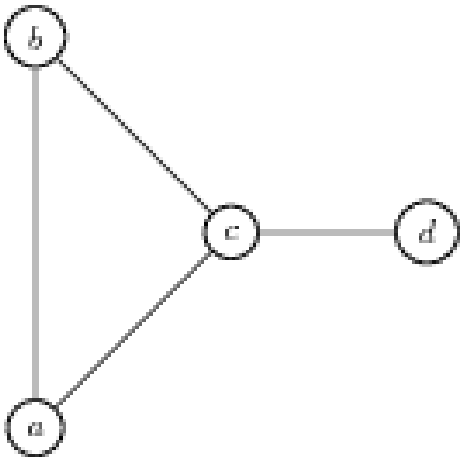
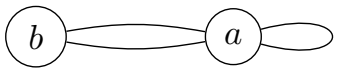



**Figure 1.2.3** The graph $K$

In the graph $K$ shown in Figure 1.2.3, vertices $a$ and $b$ have degree 2, vertex $c$ has degree 3, and vertex $d$ has degree 1. □

Note that in the definition we require $G$ to be a simple graph. The notion of degree has a few pitfalls to be careful of $G$ has loops or multiple edges. We still want to the degree $d(v)$ to match the intuitive notion of the "number of edges coming out of $v$" captured in the drawing with a small circle. The trap to beware is that this notion no longer agrees with "the number of vertices adjacent to $v$" or the "the number of edges incident to $v$"

**Example 1.2.4**
The graph $G$ to the right has two vertices, $a$ and $b$, and three edges, two between $a$ and $b$, and a loop at $a$. Vertex $a$ has degree 4, and vertex $b$ has degree 2.




□

### 1.2.2 Extended example: Chemistry

In organic chemistry, molecules are frequently drawn as graphs, with the vertices being atoms, and an edge betwen two vertices if and only if the corresponding atoms have a covalent bond between them (that is, they share a vertex).

**Example 1.2.5 Alkanes.** □

The location of an element on the periodic table determines the valency of the element -- hence the degree that vertex has in any molecule containing that graph:

- Hydrogen (H) and Fluorine (F) have degree 1
- Oxygen (O) and Sulfur (S) have degree 2
- Nitrogen (N) and Phosphorous (P) have degree 3
- Carbon (C) has degree 4

Usually, most of the atoms involved are carbon and hydrogen. Carbon atoms are not labelled with a C, but just left blank, while hydrogen atoms are left off completely. One can then complete the full structure of the molecule using the valency of each vertex. On the exam, you may have to know that Carbon has degree 4 and Hydrogen has degree 1; the valency of any other atom would be provided to you

Graphs coming from organic chemistry do not have to be simple – sometimes there are double bonds, where a pair of carbon atoms have two edges between them.

**Example 1.2.6** Saturated vs. unsaturated □

If we know the chemical formula of a molecule, then we know how many vertices of each degree it has. For a general graph, this information is known as the degree sequence

**Definition 1.2.7 Degree sequence.** The degree sequence of a graph is just the list (with multiplicity) of the degrees of all the vertices. ♢

The following sage code draws a random graph with 7 vertices and 10 edges, and then gives its degree sequence. You can tweak the code to generate graphs with different number of vertices and edges, and run the code multiple times, and the degree sequence should become clear.

```
vertices = 7
edges = 10
g = graphs.RandomGNM(vertices,edges)
```

```
g.show()
print g.degree_sequence()
```

Knowing the chemical composition of a molecule determines the degree sequence of its corresponding graph. However, it is possible that the same set of atoms may be put together into a molecule in more than one different ways. In chemistry, these are called *isomers*. In terms of graphs, this corresponds to different graphs that have the same degree sequence.

An important special case is the constant degree sequence.

**Definition 1.2.8 Regular graphs.** A graph $\Gamma$ is *d-regular*, or *regular of degree d* if every vertex $v \in \Gamma$ has the same degree $d$, i.e. $d(v) = d$. $\Diamond$

As a common special case, a regular graph where every vertex has degree three is called *trivalent*, or *cubic*.

Some quick examples:

1. The cycle graph $C_n$ is two-regular
2. The complete graph $K_n$ is $(n-1)$-regular
3. The Petersen graph is trivalent

### 1.2.3 Handshaking lemma and first applications

To motivative the Handshaking Lemma, we consider the following question. Suppose there seven people at a party. Is it possible that everyone at the party knows exactly three other people?

We can model the situation a graph, with vertices being people at the party, and an edge between two vertices if the corresponding people know each other. The question is then asking for the existence of a graph with seven vertices so that every vertex has degree three. It is then natural to attempt to solve the problem by trying to draw such a graph. After a few foiled attempts, we begin to suspect that it's not possible, but doing a case-by-case elimination of all the possibilities is daunting. It's easier to find a reason why we can't draw such a graph.

We will do this as follows: suppose that everyone at the party who knows each other shakes hands. How many handshakes will occur? On the one hand, from the definitions this would just be the number of edges in the graph. On the other hand, we can count the number of handshakes working person-by-person: each person knows three other people, and so is involved in three handshakes. But each handshake involves two people, and so if we count $7 * 3$ we've counted each handhsake twice, and so there should be $7 * 3/2 = 10.5$ handshakes happening, which makes no sense, as we can't have half a handshake. Thus, we have a contradiction, and we conclude such a party isn't possible.

Euler's handshaking Lemma is a generalization of the argument we just made to an arbitrary graph.

**Theorem 1.2.9** *(Euler's handshaking Lemma)*

$$\sum_{v \in V(G)} d(v) = 2|E(G)|$$

*Proof.* We count the "ends" of edges two different ways. On the one hand, every end occurs at a vertex, and at vertex $v$ there are $d(v)$ ends, and so the total number of ends is the sum on the left hand side. On the other hand, every edge has exactly two ends, and so the number of ends is twice the number of edges, giving the right hand side. ■

We have seen already seen one use of Euler's handshaking Lemma, but it will be particularly useful in Chapter 3, when we study graphs on surfaces.

## 1.3 Graph Isomorphisms

Generally speaking in mathematics, we say that two objects are "isomorphic" if they are "the same" in terms of whatever structure we happen to be studying. The symmetric group $S_3$ and the symmetry group of an equilateral triangle $D_6$ are isomorphic. In this section we briefly briefly discuss isomorphisms of graphs.

### 1.3.1 Isomorphic graphs

The "same" graph can be drawn in the plane in multiple different ways. For instance, the two graphs below are each the "cube graph", with vertices the 8 corners of a cube, and an edge between two vertices if they're connected by an edge of the cube:

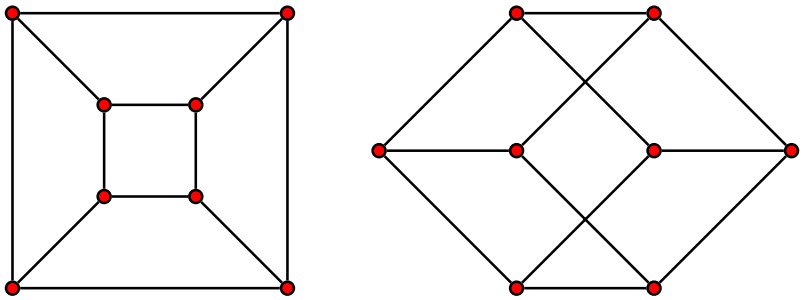


**Figure 1.3.1** Two drawings of the cube graph

**Example 1.3.2** It is not hard to see that the two graphs above are both drawings of the cube, but for more complicated graphs it can be quite difficult at first glance to tell whether or not two graphs are the same. For instance, there are many ways to draw the Petersen graph that aren't immediately obvious to be the same.

This animated gif created by Michael Sollami for this Quanta Magazine article on the Graph Isomorphism problem illustrates many different such drawings in a way that makes the isomorphisms apparent. □

**Definition 1.3.3** An isomorphism $\varphi : G \to H$ of simple graphs is a biject $\varphi : V(G) \to V(H)$ between their vertex sets that preserves the number of edges between vertices. In other words, $\varphi(v)$ and $\varphi(w)$ are adjacent in $H$ if and only if $v$ and $w$ are adjancent in $G$. ♢

**Example 1.3.4**

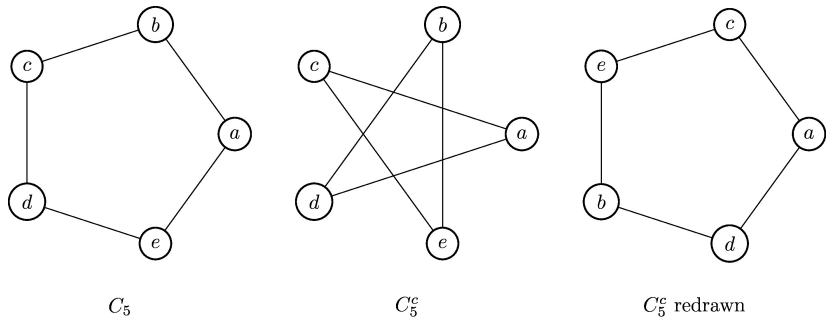



**Figure 1.3.5** $C_5$ is isomorphic to its complement $C_5^c$

The cycle graph on 5 vertices, $C_5$ is isomorphic to its complement, $C_5^c$. The cycle $C_5$ is usually drawn as a pentagon, and if we were then going to naively draw $C_5^c$ we would draw a 5-sided star. However, we could draw $C_5^c$ differently as shown, making it clear that it is isomorphic to $C_5$, with isomorphism $\varphi : C_5 \to C_5^c$ defined by $\varphi(a) = a, \varphi(b) = c, \varphi(c) = e, \varphi(d) = b, \varphi(e) = d$. □

Although solving the graph isomorphism problem for general graphs is quite difficult, doing it for small graphs by hand is not too bad and is something you must be able to do for the exam. If the two graphs are actually isomorphic, then you should show this by exhibiting an isomrophism; that is, writing down an explicit bijection between their vertex sets with the desired properties. The most attractive way of doing this, for humans, is to label the vertices of both copies with the same letter set.

If two graphs are not isomorphic, then you have to be able to prove that they aren't. Of course, one can do this by exhaustively describing the possibilities, but usually it's easier to do this by giving an obstruction – something that is different between the two graphs. One easy example is that isomorphic graphs have to have the same number of edges and vertices. We'll discuss some others in the next section

### 1.3.2 Heuristics for showing graphs are or aren't isomorphic

Another, only slightly more advanced invariant is the degree sequence of a graph that we saw last lecture in our discussion of chemistry.

If $\varphi : G \to H$ is an isomorphism of graphs, than we must have $d(\varphi(v)) = d(v)$ for all vertices $v \in G$, and since isomorphisms are bijections on the vertex set, we see the degree sequence must be preserved. However, just because two graphs have the same degree sequences does not mean they are isomorphic.

Slightly subtler invariants are number and length of cycles and paths.

### 1.3.3 Cultural Literacy: The Graph Isomorphism Problem

This section, as all "Cultural Literacy" sections, is information that you may find interesting, but won't be examined.

The graph isomorphism problem is the following: given two graphs $G$ and $H$, determine whether or not $G$ and $H$ are isomorphic. Clearly, for any two graphs $G$ and $H$, the problem is solvable: if $G$ and $H$ both of $n$ vertices, then there are $n!$ different bijections between their vertex sets. One could simply examine each vertex bijection in turn, checking whether or not it maps edges to edges.

The problem is interesting because the naive algorithm discussed above is not very efficient: for large $n$, $n!$ is absolutely huge, and so in general this

algorithm will take a long time. The question is, is there are a faster way to do check? How fast can we get?

The isomorphism problem is of fundamental importance to theoretical computer science. Apart from its practical applications, the exact difficulty of the problem is unknown. Clearly, if the graphs are isomorphic, this fact can be easily demonstrated and checked, which means the Graph Isomorphism is in NP.

Most problems in NP are known either to be easy (solvable in polynomial time, P), or at least as difficult as any other problem in NP (NP complete). This is not true of the Graph Isomorphism problem. In November of last year, Laszlo Babai announced a quasipolynomial-time algorithm for the graph isomorphism problem – you can read about this work in this great popular science article.

## 1.4 Instant Insanity

So far, our motivation for studying graph theory has largely been one of philosophy and language. Before we get too much deeper, however, it may be useful to present a nontrivial and perhaps unexpected application of graph theory; an example where graph theory helps us to do something that would be difficult or impossible to do without it.

### 1.4.1 A puzzle

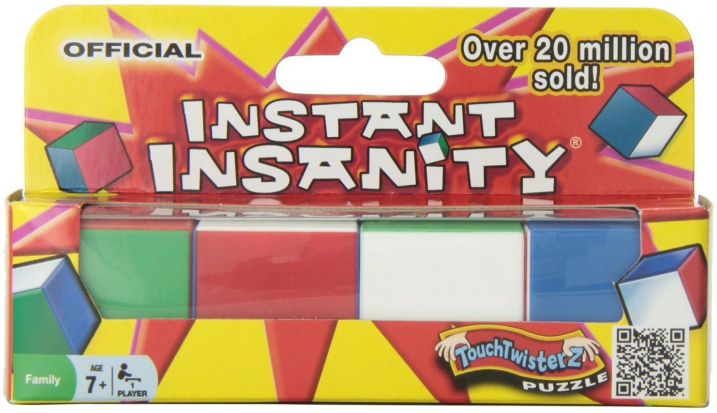



**Figure 1.4.1** Instant Insanity Package

There is a puazzle marketed under the name "Instant Insanity", one version of which is shown above. The puzzle is sometimes called "the four cubes problem", as it consists of four different cubes. Each face of each cube is painted one of four different colours: blue, green, red or yellow. The goal of the puzzle is to line the four cubes up in a row, so that along the four long edges (front, top, back, bottom) each of the four colours appears eactly once.

Depending on how the cubes are coloured, this may be not be possible, or there may be many such possibilities. In the original instant insanity, there is exactly one solution (up to certain equivalences of cube positions). The point of the cubes is that there are a large number of possible cube configurations, and so if you just look for a solution without being extremely systematic, it is highly unlikely you will find it.

But trying to be systematic and logical about the search directly is quite difficult, perhaps because we have problems holding the picture of the cube in

our mind. In what follows, we will introduce a way to translate the instant insanity puzzle into a question in graph theory. This is obviously in no way necessary to solve the puzzle, but does make it much easier. It also demonstrates the real power of graph theory as a visualization and thought aid.

There are many variations on Instant Insanity, discussions of which can be found here and here. There's also a commercial for the game.

### 1.4.2 Enter graph theory

It turns out that the important factor of the cubes is what color is on the opposite side of each face. Suppose we want face one facing forward. Then we have four different ways to rotate the cube to keep this the same. The same face will always appear on the opposite side, but we can get any of the remaining four faces to be on top, say.

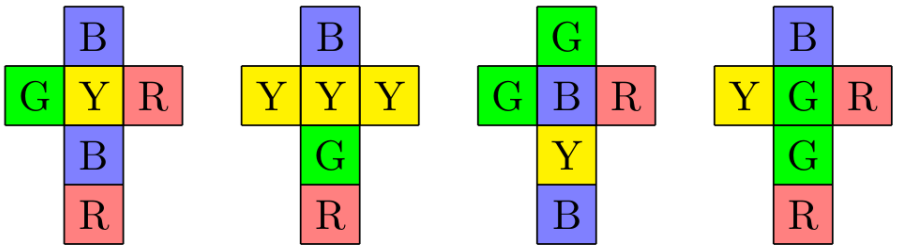



**Figure 1.4.2** An impossible set of cubes

Let us encode this information in a graph. The vertices of the graph will be the four colors, B, G, R and Y. We will put an edge between two colors each time they appear as opposite faces on a cube, and we will label that edge with a number 1-4 denoting which cube the two opposite faces appear. Thus, in the end the graph will have twelve edges, three with each label 1-4. For from the first cube, there will be a loop at B, and edge between G and R, and an edge between Y and R. The graph corresponding to the four cubes above is the following:

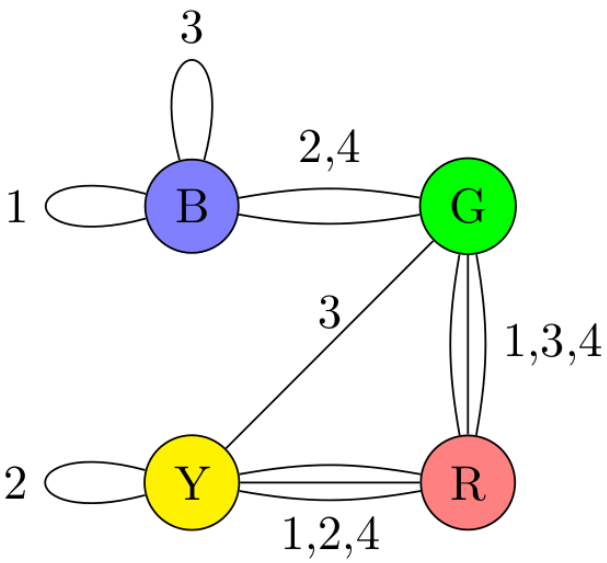



**Figure 1.4.3** The graph constructed from the cubes in Figure 1.4.2

### 1.4.3 Proving that our cubes were impossible

We now analyze the graph to prove that this set of cubes is not possible.

Suppose we had an arrangement of the cubes that was a solution. Then, from each cube, pick the edge representing the colors facing front and back on that cube. These four edges are a subgraph of our original graph, with one edge of each label, since we picked one edge from each cube. Furthermore, since we assumed the arrangement of cubes was a solution of instant insanity, each color appears once on the front face and once on the back. In terms of our subgraph, this translates into asking that each vertex has degree two.

We can get another subgraph satisfying these two properties by looking at the faces on the top and bottom for each cube and taking the corresponding edges. Furthermore, these two subgraphs do not have any edges in common.

Thus, given a solution to the instant insanity problem, we found a pair of subgraphs $S_1, S_2$ satisfying:

1. Each subgraph $S_i$ has one edge with each label 1,2,3,4
2. Every vertex of $S_i$ has degree 2
3. No edge of the original graph is used in both $S_1$ and $S_2$

As an exercise, one can check that given a pair of subgraphs satisfying 1-3, one can produce a solution to the instant insanity puzzle.

Thus, to show the set of cubes we are currently examining does not have a solution, we need to show that the graph does not have two subgraphs satisfying properties 1-3.

To do, this, we catalog all graphs satisfying properties 1-2. If every vertex has degree 2, either:

1. Every vertex has a loop
2. There is one vertex with a loop, and the rest are in a triangle
3. There are two vertices with loops and a double edge between the other two vertices

4. There are two pairs of double edges
5. All the vertices live in one four cycle
6. A subgraphs of type 1 is not possible, because G and R do not have loops.

For subgraphs of type 2, the only triangle is G-R-Y, and B does have loops. The edge between Y-G must be labeled 3, which means the loop at B must be labeled 1. This means the edge between G and R must be labeled 4 and the edge between Y and R must be 2, giving the following subgraph:

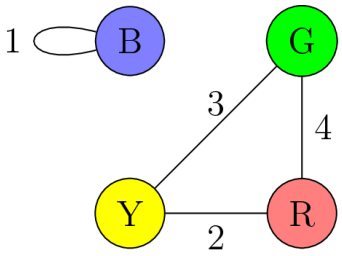



**Figure 1.4.4** A subgraph for a solution for one pair of faces

For type 3, the only option is to have loops at B and Y and a double edge between G and R. We see the loop at Y must be labeled 2, one of the edges between G and R must be 4, and the loop at B and the other edge between G and R can switch between 1 and 3, giving two possibilities:

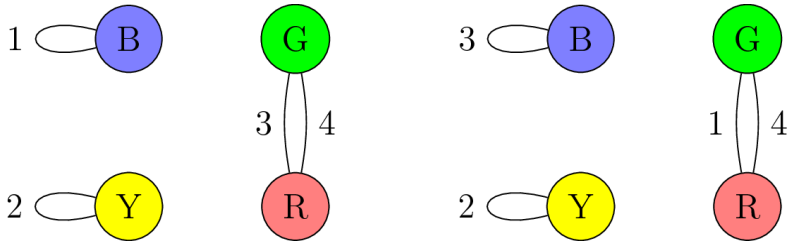



**Figure 1.4.5** Two more subgraphs for a partial solutions

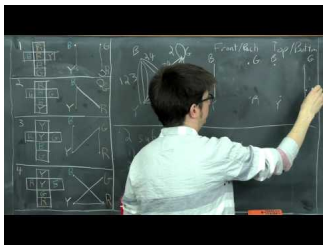
For subgraphs of type 4, the only option would be to have a double edge between B and G and another between Y and R; however, none of these edges are labeled 3 and this option is not possible.

Finally, subgraphs of type 5 cannot happen because B is only adjacent to G and to itself; to be in a four cycle it would have two be adjacent to two vertices that aren't itself.

This gives three different possibilities for the subgraphs SiSi that satisfy properties 1 and 2. However, all three possibilities contain the the edge labeled 4 between G and R; hence we cannot choice two of them with disjoint edges, and the instant insanity puzzle with these cubes does not have a solution.

### 1.4.4 Other cube sets

The methods above also give a way to find the solution to a set of instant insanity cubes should one exist. I illustrate this in the following Youtube video:



YouTube: https://www.youtube.com/watch?v=GsbhRfjaaN8

## 1.5 Trees

A very important class of graphs are trees -- they are connected, but just barely: removing any edge causes them not to be connected.

### 1.5.1 Basics on trees

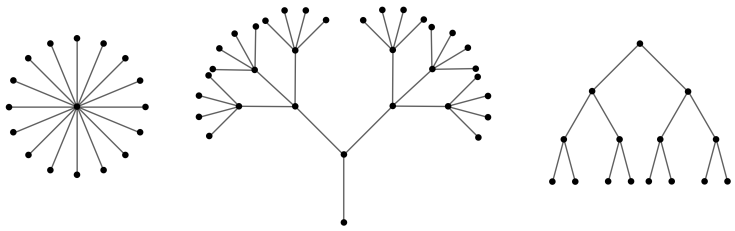


**Figure 1.5.1** A forest consisting of three trees

The figure above shows some examples of the trees. Meanwhile, the cycle graph $\mathbf{C}_n$ or the complete graph $\mathbf{K}_n$ with $n \geq 3$ are not trees: we can remove an edge from these graphs and they'd still be connected. The formal definition of a tree is as follows:

**Definition 1.5.2 Trees and Forests.** A graph $\mathbf{G}$ is a *tree* if:

1. $\mathbf{G}$ is connected
2. $\mathbf{G}$ has no cycles

A not necessarily connected graph with no cycles is called a *forest*, so that a forest is a union of trees. ◊

Informally, the condition that $\mathbf{G}$ is connected is asking that $\mathbf{G}$ have enough edges, while the condition that $\mathbf{G}$ has no cycles is asking for $\mathbf{G}$ not to have too many edges. Thus, trees are sort of "goldilocks" graphs -- they have enough edges, but not too many -- they're connected, but just barely.

The following Theorem gives many alternate characterisations of trees, and makes more precise the intuition of trees as "goldilocks graphs".

**Theorem 1.5.3** *Let $\mathbf{G}$ be a graph with $n$ vertices. The following are equivalent:*

1. *$\mathbf{G}$ is a tree.*
2. *Between any two vertices $a, b \in V(\mathbf{G})$, there is a unique path.*
3. *$\mathbf{G}$ is connected, but removing any edge makes $\mathbf{G}$ disconnected.*
4. *$\mathbf{G}$ has no cycles, but adding any edges to $\mathbf{G}$ creates a cycle.*
5. *$\mathbf{G}$ is connected and has $n-1$ edges*
6. *$\mathbf{G}$ has no cycles and has $n-1$ edges*

We will not use most of , and will not prove that all options are equivalence. We briefly proof that 1 is equivalent to 2 now, and in the next section we will carefully prove that 1 is equivalent to 5, as we will use this fact a few times. The rest make a good exercise to check your basic understanding.

To see that 1 and 2 are equivalent, note that being connected by definition means there is a path between every two vertices. As a tree is a connected graph without any cycles, to finish seeing 1 and 2 are equivalent is exactly Lemma 1.5.5, whose main idea is contained in Figure 1.5.4

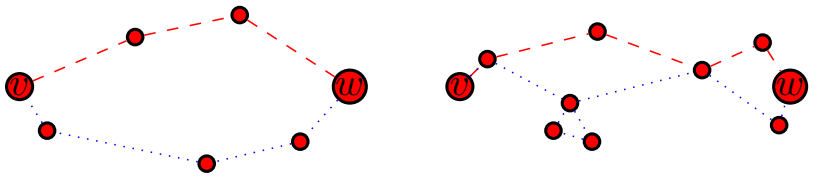



**Figure 1.5.4** Two different paths create a cycle

**Lemma 1.5.5** *A graph* **G** *has no cycles if and only if there is at most one path between any two vertices of* **G**.

*Proof.* If **G** has a cycle, the there are at least two paths between any vertex on that cycle -- the paths going each way around the cycle. Thus, we just have to show that if there there are two paths between between $v$ and $w$, then **G** has a cycle.

In the easy case, the two paths will contain no vertices in common except for $v$ and $w$, and so the union of the two paths will be a cycle. In general, the paths will share other vertices and edges -- they may well repeat vertices and edges themselves. But in any case, by considering some subset of these two paths will be a cycle. ■

### 1.5.2 Leaves

To prove Theorem , we first need to introduce the concept of leaves.

**Definition 1.5.6 Leaf.** A vertex $v \in \mathbf{G}$ is called a *leaf* if it has degree one, i.e. if $d(v) = 1$ ◊

When looked at a drawn graph, this definition is fairly intuitive: real life trees branch out and split in leaves, just like mathematical trees.

**Lemma 1.5.7 Trees have leaves.** *Let* **T** *be a finite tree with at least two vertices. Then* **T** *has at least two leaves.*

*Proof.* By assumption, **T** has at least two vertices, say $v_0$ and $w_0$. Since **T** is a tree it is connected, and so in particular there must be a path between $v_0$ and $w_0$; let $v_i$ be the vertices in this path, and let $e_i$ be the edge in the path joining $v_{i-1}$ to $v_i$.

Since $v_m$ is adjacent to $e_m$ it has degree at least one; if it has degree 1 it is a leaf, and we've found a leaf. If $v_m$ is not a leaf, then there must be another edge coming out of it, say $e_{m+1}$ going to $v_{m+1}$. Note that $v_{m+1}$ cannot be any of the vertices we've already found, as then we'd have more than one path between two vertices and hence a loop, but **T** was a tree. Thus we can make the path a bit longer.

We can now continue this argument inductively as long as the vertex at the of the path has degree higher than 1. But since **T** is finite, and we never return to a vertex we've already visited, we this process must eventually terminate, but the only way this can happen is if the end vertex of the path has degree 1, that is, if it's a leaf.

A similar argument extending from $v_0$, the other end of the path, shows

that we must eventually reach a different leaf from that end, and so **T** must have at least two leaves, as desired. ■

Now that we have a basic understanding of leaves, we are ready to prove the following:

**Lemma 1.5.8** *A simple graph* **G** *with* $n$ *vertices is a tree if and only if is connected and has* $n-1$ *edges.*

*Proof.* Since being connected is half of the requirement of being a tree, we need to show that a connected graph on $n$ vertices is a tree if and only if it has $n-1$ edges.

Let us first prove that a tree on $n$ vertices has $n-1$ edges. We will use induction on $n$. As bases, there is only one tree with 1 vertex, and it does in fact have 0 edges, and there is only one tree with two vertices, and it does in fact have 1 edge. So for the inductive step, let us suppose that we know that all trees with $k < n$ vertices have $k-1$ edges, and let **T** be a tree with $n$ vertices. By Lemma, we know that **T** has a leaf $v$, which by the definition of leaves is connected to the rest of **T** by a single edge $e$. If we delete $v$ and $e$ from **T**, we get a smaller graph $\mathbf{T}'$ ince which has one less vertex and one less edge than **T** did.

Since **T** was a tree, it follows that $\mathbf{T}'$ is a tree, too -- check this yourself, using the definition of a tree! Then, since $\mathbf{T}'$ is a tree with $n-1$ vertices, byt the inductive hypothesis it follows that it has one less edge $n-2$ edges, and so **T** must have $n-1$ edges, which is what we were trying to show.

Now we show the other direction: that if **G** is a simple connected graph with $n$ vertices and $n-1$ edges, then **G** is a tree. The basic structure of the proof is the same as the other direction: find a vertex $v$ adjacent to a single edge $e$, and delete $v$ and $e$ to get a smaller tree, where we can assume the proposition holds, and then use induction.

To play the role of the lemma that every tree has a leaf, we will establish the following statement: if **G** is a connected graph with $n$ vertices and $n-1$ edges, then /p>**G** has a vertex of degree 1. Note that since **G** is connected, it can't have any vertices of degree 0, and so to prove it has a vertex of degree 1 it is enough to show that it has a vertex of degree strictly less than 2. Thus, for contradiction assume that every vertex $v$ of **G** has degree $d(v) \geq 2$. But then the handshaking lemma would say:

$$2(n-1) = 2e = \sum d(v) \geq \sum 2 = 2n$$

a contradiction, and hence **G** must have a vertex of degree 1, as desired. ■

### 1.5.3 Trees in Chemistry

Our brief study of trees has the following consequence for chemistry: whether or not a molecule is a tree is determined just by its chemical formula, and not how it's put together. Equivalently, if one isomer of a molecule is a tree, then all isomers of the molecule are.

The argument runs as follows. Knowing the chemical formula of a molecule means we know the degree sequence of the corresonding graphs. Molecules are by definition connected graphs, so to be a tree it is enough to show that the graph has one less edge than the number of vertices. But we can compute the number of edges from the degree sequence by using the Handshaking Lemma.

**Example 1.5.9 Alkanes are trees.** Any molecule with formula $C_nH_{2n+2}$ is an *alkane*. Although in general alkanes can have multiple isomers, every isomer of an Alkane will always be a tree, as we now show.

To show a graph is a tree, it suffices to show that it is connected and that the number of edges is one less than the number of vertices. Since Alkanes are molecules, we know that the graph is connected. Furthermore, $C_nH_{2n+2}$ has $n$ carbons adn $2n+2$ hydrogens, and hence has $3n+2$ vertices. Thus, it is enough to show that any molecule with formula $C_nH_{2n+2}$ has $3n+1$ edges.

To do this, we use the handshaking lemma: $2e = \sum d(v)$. Each of the $n$ carbons has degree 4, so the carbons contribute $4n$ to the total degree, and each hydrogen of the $2n+2$ has degree 1 and so only contributes 1 to the sum of degrees. Hence, twice the number of edges is equal to $4n + (2n+2) = 6n+2$, and so there are $3n+1$ edges, as desired. □

As an early application of graph theory, Cayley used these ideas to count the number of isomers of Alkanes (with some mistakes). In general, you can count isomers of any molecule by counting isomorphism classes of graphs with given degree sequences, but it can help organize the search to know, e.g., that they're all trees. To make sure we don't miss or double count any, it's useful to organize the enumeration according to some principle; for Alkanes one way is to organize according to the longest path of carbons, another is to restrict degree sequences to just how the carbons have connected to other carbons.

**Example 1.5.10 Counting isomers of $C_6H_{14}$.** We illustrate these both of these methods. We first illustrate by length of the longest path of carbons.

- Chain length six: Since we've used all carbons then there are no more choices, and there is only one such isomer.
- Chain length five: We need to add one more carbon. We can't add it to either of the end carbons, because then we'd have a path of length 6. We can add it to the central of the two chains, or alternatively to one either side of central -- the last two trees are isomorphic, giving us two possibilities
- Chain length four: We need to add two more carbons. Again, they can't be added to either of the end carbons, or we'd have a longer chain length. Therefore, they most be added to the two central carbons. One case is that we add one carbon to each of the two central carbons. Alternatively, we could add both the carbons to the same "central" carbon reversing the order of the chain is a symmetry that interchanges the two central atoms. We could add each carbon directly to the existing carbon in the chain, or we could add them one after the other making a chain of length two. However, the resulting graph would have a chain of length 5 and already be counted. Thus, there are two possibilities here.
- Chain length three: We need at add three more carbons, and there's only one central carbon to attach them to. We can't add them all directly to this central carbon, as that would create a carbon of degree 5. On the other hand, once we add a chain of length longer than one to this central carbon we'd have a path of length 4 or greater.

Thus, we see there are five isomers of $C_6H_{14}$. Alternatively, we could organize our search by deleting the hydrogens, and then considering the degrees of the resulting carbon-carbon graph.

- Degree at most two: If the resulting tree only had carbons of degree at most two, then it would have to be the path graph $P_6$, and so we only have one possibility here.
- One vertex of degree three: If the resulting graph had exactly one carbon of degree three, that vertex and its three neighbours would account for 4

of our 6 carbons, and so we'd have to add two more. We couldn't add them directly to the same vertex, as that would create a second vertex of degree three. So, they could either be added as a chain of length two to one of the leaves of the existing graph, or they could be added to two separate leaves. Drawing these graphs we see they're not isomorphic, and so we have two possibilities here.

- Two vertices of degree three: If we have two vertices of degree three, one sees they'd have to be adjacent to each other, resulting in one possibity.
- Vertex of degree four: A vertex of degree 4 and its four neighbours would account for all but one of the carbons. We could add that carbon to any of the leaves, and get one more possiblity.

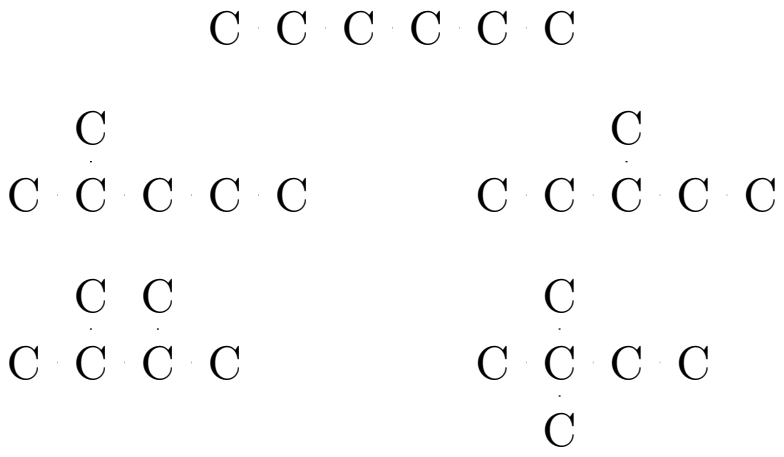


**Figure 1.5.11** Isomers of $C_6H_{12}$

Since carbons only have degree 4, the tree with six vertices where all are connected to a central vertex isn't allowed, and we have found all the isomers. □

## 1.6 Exercises

**1.** For each of the following sequences, either give an example of such a graph, or explain why one does not exist.

(a) A graph with six vertices whose degree sequence is $[5, 5, 4, 3, 2, 2]$

(b) A graph with six vertices whose degree sequence is $[5, 5, 4, 3, 3, 2]$

(c) A graph with six vertices whose degree sequence is $[5, 5, 5, 5, 3, 3]$

(d) A simple graph with six vertices whose degree sequence is $[5, 5, 5, 5, 3, 3]$

**2.** For the next Olympic Winter Games, the organizers wish to expand the number of teams competing in curling. They wish to have 14 teams enter, divided into two pools of seven teams each. Right now, they're thinking of requiring that in preliminary play each team will play seven games against distinct opponents. Five of the opponents will come from their own pool and two of the opponents will come from the other pool. They're having trouble setting up such a schedule, so they've come to you. By using an appropriate graph-theoretic model, either argue that they cannot use their current plan or devise a way for them to do so.

**3.** Figure 1.6.1 contains four graphs on six vertices. Determine which (if any) pairs of graphs are isomorphic. For pairs that are isomorphic, give an isomorphism between the two graphs. For pairs that are not isomorphic, explain why.

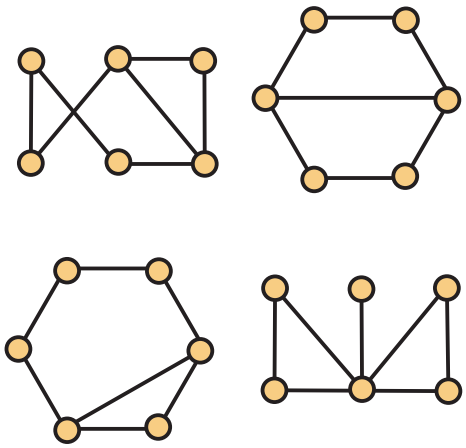


**Figure 1.6.1** Are these graphs isomorphic?

**4.** Let **G** be a simple graph with $n$ vertices and degree sequence $a_1, a_2, \ldots, a_n$. What's the degree sequence of its complement $\mathbf{G}^c$?

**5.** Let $G$ be the graph with graph with vertices consisting of the 10 three element subsets of $\{a, b, c, d, e\}$, and two vertices adjacent if they share exactly one element. So, for example, the two vertices $v = \{a, c, e\}$ and $w = \{b, c, d\}$ are adjacent, but neither $v$ or $w$ is adjacent to $u = \{a, b, c\}$.

Draw $G$ in a way that shows it is isomorphic to the Petersen graph.

Now let $H$ be the graph with vertices consisting of the 10 *two* element subsets of $\{a, b, c, d, e\}$, and two vertices adjacent if they share *no* elements. Without drawing $H$, write down an isomorphism between $G$ and $H$. Hint: There's a "natural" bijection between the two and three element subsets of $\{a, b, c, d, e\}$

**6.** Recall that $G^c$ denotes the complement of a graph $G$. Prove that $f : G \to H$ is an isomorphism of graphs if and only if $f : G^c \to H^c$ is an isomorphism.

**7.** Determine the number of non-isomorphic simple graphs with seven vertices such that each vertex has degree at least five.

**Hint**. Consider the previous exercise

**8.** Consider the standard Instant Insanity puzzle, with four cubes and four colours. Explain why one would expect there to be 331,776 different cube configurations. Further explain why there would be fewer configurations if any cubes are coloured with symmetries.

In the text, we solve the puzzle by finding certain pairs of subgraphs. Assuming that none of the cubes are coloured symmetrically, explain why each pair of subgraphs corresponds to at least 8 different cube configurations that are actually solutions, and why, depending on the isomorphism type of the subgraphs found, there may be more solutions.

**9.** Variations of the Insant Insanity puzzle increase the number of cubes and the number of colours. Explain how to modify our graph theoretic solution to solve the puzzle when we have $n$ cubes, each face of which is coloured one of $n$ colours, and we want to line up the cubes so that each of the top, bottom, front and rear strings of cubes displays each of the $n$ colours exactly once.

**10.** Use the method from the previous question to solve the following set of six cubes, marketed under the name "Drive ya crazy", where each face is coloured either blue, cyan, green, orange, red, or yellow.

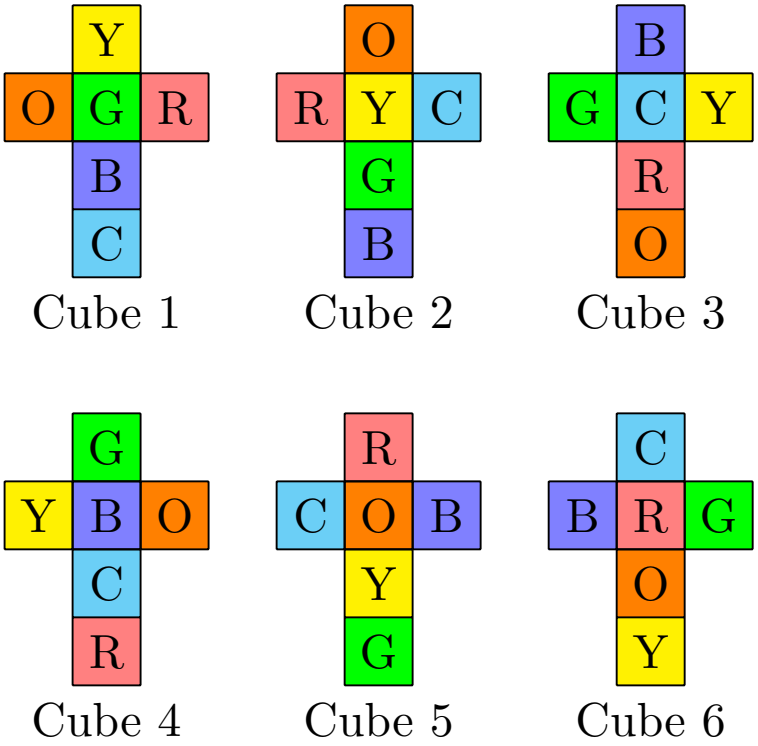



**Figure 1.6.2** The six cubes from "Drive Ya crazy"

# Chapter 2

# Walks

In this chapter we investigate walks in graphs. We first look at some basic definitions and examples, we discuss Dijkstra's algorithm for finding the shortest path between two points in a weighted graph, and we discuss the notions of Eulerian and Hamiltonian graphs.

## 2.1 Walks: the basics

If the edges in a graph $\Gamma$ represent connections between different cities, it is obvious to strart planning longer trips that compose several of these connections. The notion of a *walk* formally captures this definition; the formal notions of *path* and *trail* further ask that we not double back on ourselves or repeat ourselves in certain formally defined ways.

Once we've done that, we investigate what it means for a graph to be connected or disconnected.

### 2.1.1 Walks and connectedness

Before we see the formal definition of a walk, it will be useful to see an example:

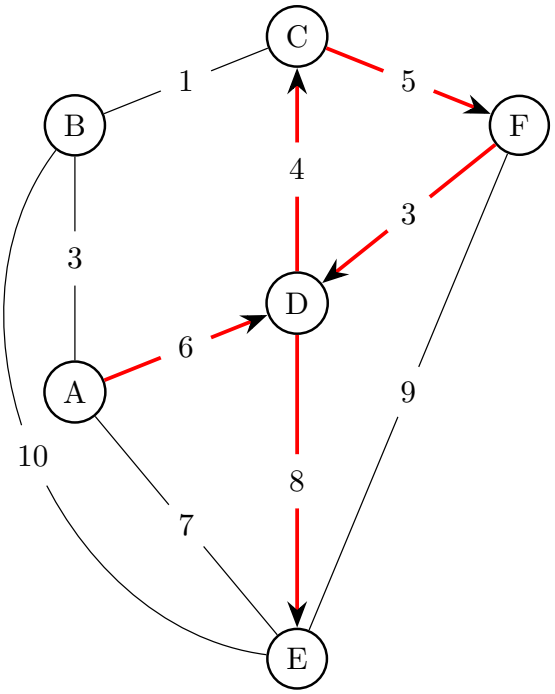



**Figure 2.1.1** Example of a walk

In the graph shown, the vertices are labelled with letters, and the edges are labelled with numbers, and we have a walk highlighted in red, and with arrowtips drawn on the edges. Starting from vertex $A$, we can take edge 6 to vertex $D$, and then edge 5 to vertex $C$, edge 5 to vertext $F$, edge 3 back to vertex $D$, and finally edge 8 to vertex $E$. Intuitively,then, a walk strings together several edges that share vertices in between. Makign that formal, we have the following.

**Definition 2.1.2 Walk.** *walk* in a graph $\Gamma$ is a sequence

$$v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n$$

where the $v_i$ are vertices, the $e_j$ are edges, and the edge $e_j$ goes between vertices $v_{j-1}$ and $v_j$.

We say that the walk is between vertices $a = v_0$ and $b = v_n$ ◊

With this notation for a walk, Example Figure 2.1.1, the walk shown would be written $A, 6, D, 4, C, 5, F, 3, D, 8, E$. The visual representation of the walk on the graph is vastly more intuitive, the written one feeling cumbersome in comparison.

The definition of walk above contains some extra information. If we just know the sequence of edges we can reconstruct what the vertices have to be (assuming we have at least two edges in the walk). Alternatively, if the graph $\Gamma$ does not have multiple edges, it is enough to just know the vertices $v_i$, but if $\Gamma$ has multiple edges that just knowing the vertices does not determine the walk.

**Definition 2.1.3 Connected.** We say a graph $\Gamma$ is *connected* if for any two vertices $v, w$, there is a walk from $v$ to $w$ in $\Gamma$. ◊

**Definition 2.1.4 Disjoint union.** Given two graphs $\Gamma_1$ and $\Gamma_2$, the *disjoint union* $\Gamma_1 \sqcup \Gamma_2$ is obtained by taking the disjoint union of both the vertices and edges of $\Gamma_1$ and $\Gamma_2$. So $\Gamma_1 \sqcup \Gamma_2$ consists of a copy of $\Gamma_1$ and a copy of $\Gamma_2$, with no edges in between the two graphs. ◊

**Definition 2.1.5 Disconnected.** A graph $\Gamma$ is *disconnected* if we can write $\Gamma = \Gamma_1 \sqcup \Gamma_2$ for two proper (i.e., not all of $\Gamma$) subgraphs $\Gamma_1$ and $\Gamma_2$. ◊

We now have a definition for what it means for a graph to be connected, and another for what it means for a graph to be disconnected. From everday usage of this words, we would certainly hope that a graph is disconnected if and only if it is not connected. However, it is not immediately clear from the definitions as written that this is the case.

**Lemma 2.1.6** *The following are equivalent:*

1. . $\Gamma$ *is connected*

2. $\Gamma$ *is not disconnected*

*Proof.* 1 implies 2: Supppose that $\Gamma$ is connected, and let $v, w \in V(\Gamma)$; we want to show that there is a walk from $v$ to $w$.

Define $S \subset V(\Gamma)$ to be the set of all vertices $u \in V(\Gamma)$ so that there is a walk from $v$ to $u$; we want to show that $w \in S$.

First, observe that there are no edges from $S$ to $V(\Gamma) \setminus S$. Suppose that $e$ was an edge between $a \in S$ and $b \in \Gamma \setminus S$. Since $a \in S$, by the definition of $S$ there is a walk $v = v_0 v_1 v_2 \cdots v_m = a$ from $v$ to $a$. We can add the edge $e$ to the end of the walk, to get a walk from $v$ to $b$, and hence by definition $b \in S$.

Now suppose that $w \notin S$. Then $S$ and $V(\Gamma) \setminus S$ are both nonempty, and by the above there are no edges between them, and so $\Gamma$ is not connected, a contradiction.

To prove 2 implies 1, we prove the contrapositive. If $\Gamma$ is not connected, then there are two vertices $v, w \in V(\Gamma)$ so that there is no walk from $v$ to $w$.

Suppose that $\Gamma = \Gamma_1 \sqcup \Gamma_2$, and pick $v \in V(\Gamma_1), w \in V(\Gamma_2)$. Any walk from $v$ to $w$ starts in $V(\Gamma_1)$ and ends in $V(\Gamma_2)$, and so at some point there must be an edge from a vertex in $\Gamma_1$ to a vertex in $\Gamma_2$, but there are no such edges □ ■

### 2.1.2 Types of Walks

Many questions in graph theory ask whether or not a walk of a certain type exists on a graph: we introduce some notation that will be needed for these questions.

**Definition 2.1.7** We say a walk is *closed* if it starts and ends on the same vertex; i.e., $v_0 = v_n$. The *length* of a walk is $n$, the number of edges in it. The *distance* between two vertices $v$ and $w$ is the length of the shortest walk from $v$ to $w$, if one exists, and $\infty$ if one does not exist. ◊

It is sometimes convenient to have terminology for walks that don't backtrack on themselves:

**Definition 2.1.8**

1. If the edges $e_i$ of the walk are all distinct, we call it a *trail*

2. If the vertices $v_i$ of the walk are all distinct (except possibly $v_0 = v_m$), we call the walk a *path*. The exception is to allow for the possibility of closed paths.

◊

**Lemma 2.1.9** *Let $v, w \in V(\Gamma)$. The following are equivalent:*

1. *There is a walk from $v$ to $w$*

2. *There is a trail from $v$ to $w$*

3. *There is a path from $v$ to $w$.*

As is often the case, the formal write-up of the proof makes something that can seem very easy intuitively look laborious, so it's worth anlysing it briefly for our example walk $A - D - C - F - D - E$ from Figure 2.1.1. This walk is not a path as it repeats the vertex $D$; however, we may simply remove the triangle $D - C - F - D$ from the walk to get the trail $A - D - E$. this idea is what works in general.

*Proof.* It is immediate from the definitions that 3 implies 2 which implies 1, as any path is a trail, and any trail is a walk.

That 1 implies 3 is intuitively obvious: if you repeat a vertex, then you've visited someplace twice, and weren't taking the shortest route. Let's make this argument mathematically precise.

Suppose we have a walk $v = v_0, e_1, \ldots, e_m, v_m = w$ that is not a path. Then, we must repeat some vertex, say $v_i = v_k$, with $i < k$. Then we can cut out all the vertices and edges between $v_i$ and $v_k$ to obtain a new walk

$$v = v_0, e_1, v_1, \ldots, e_i, v_i = v_k, e_{k+1}, v_{k+1}, e_{k+2}, v_{k+2}, \ldots, v_m$$

Since $i < k$, the new walk is strictly shorter than our original walk. Since the length of a walk is finite, if we iterate this process the result must eventually terminate. That means all our vertices are distinct, and hence is a path. ■

## 2.2 Eulerian Walks

In this section we introduce the problem of Eulerian walks, often hailed as the origins of graph theroy. We will see that determining whether or not a walk has an Eulerian circuit will turn out to be easy; in contrast, the problem of determining whether or not one has a Hamiltonian walk, which seems very similar, will turn out to be very difficult.

### 2.2.1 The bridges of Konigsburg

The city of Konigsberg (now Kaliningrad) was built on two sides of a river, near the site of two large islands. The four sectors of the city were connected by seven bridges, as follows (picture from Wikipedia):

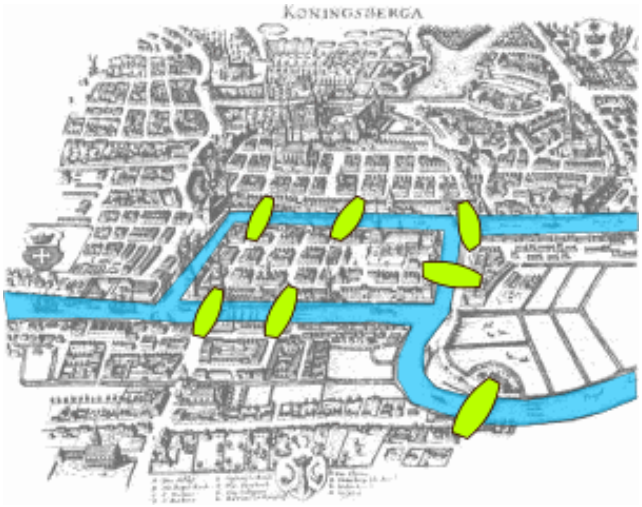



**Figure 2.2.1** The city of Konigsburg in Euler's time

A group of friends enjoyed strolling through the city, and created a game: could they take a walk in the city, crossing every bridge exactly once, and return to where they started from? They couldn't find such a walk, but they couldn't prove such a walk wasn't possible, and so they wrote to the mathematician Euler, who proved that such a walk is not possible.

### 2.2.2 Eulerian Walks: definitions

We will formalize the problem presented by the citizens of Konigsburg in graph theory, which will immediately present an obvious generalization.

We may represent the city of Konigsburg as a graph $\Gamma_K$; the four sectors of town will be the vertices of $\Gamma_K$, and edges between vertices will represent the bridges (hence, this will not be a simple graph).

Then, the question reduces to finding a closed walk in the graph that will uses every edge *exactly* once. In particular, this walk will not use any edge more than once and hence will be a trail.

**Checkpoint 2.2.2**

**Definition 2.2.3** Let $G$ be a graph. An *Eulerian cycle* is a closed walk that uses every edge of $G$ exactly once.

If $G$ has an Eulerian cycle, we say that $G$ is *Eulerian.*

If we weaken the requirement, and do not require the walk to be closed, we call it an Euler path, and if a graph $G$ has an Eulerian path but not an Eulerian cycle, we say $G$ is *semi-Eulerian* ◊

The question of the walkers of Konigsburg is then equivalent to asking if the graph $\Gamma_K$ is Eulerian. The birth of graph theory is usually marked to the following theorem, proven by Euler:

**Theorem 2.2.4** *A connected graph* $\Gamma$ *is Eulerian if and only if every vertex of* $\Gamma$ *has even degree*

### 2.2.3 A digression on proofs, formality, and intuition

Before discussing the proof of Theorem 2.2.4, it's worth a little meta-discussion about proofs, intuition vs. rigor, and mathematics as a whole. The proofing Theorem 2.2.4 is a common exam question, and you may not be used to studying for reproducing proofs on exams. Certainly one way to prepare for such a question is to memorize the proof word for word. There doesn't seem to be a lot of obvious value in this approach, however. So why ask these questions on the exam? And this opens the door to more philosophical questions as well: how should we think/interact with proofs anyway? What's the point of it all?

Usually in books or in lectures, proofs are only given in slick, elegant, polished formal versions. There are many reasons for this: there's a certain beauty to it; it's important to write it out formally to make sure it's all correct; there's only so much time in lectures, and brevity is a virtue anyway. People turn away from long works, and the main point of a proof, after all, is to prove something, and it's easier to check that it's all correct if it's shorter.

But there's a very real downside to this presentation of proofs as the finished, elegant thing. Most important to me is that the way mathematics is written formally on the page is very different from how it lives actively in our brains (or my brain, at least). Nobody (or certainly very few people) comes up with proofs in the elegant short start to finish way that they're written. Typically, there's a mess of chaotic half ideas that slowly get refined down to the written proof that you see. But often the mess is the exciting part,

We sketch a few of the main ideas of the proof in an informal setting now, before giving a complete formal proof. To learn this proof for the exam, you should have this informal picture in your head, and perhaps a skeleton outline of the main formal points that need to be shown. You shouldn't try to memorize the formal proof word for word like a poem; instead, practice expanding out from the informal ideas/skeleton proof to the full formal proof on your own a few times.

It is much easier to see that if a graph $G$ *is* Eulerian, then every vertex has even degree. Suppose we wanted to show that a given vertex $v$ was Eulerian; let us stand at the vertex $v$ and have a friend trace out the Eulerian cycle. We'll wait for a while, and then the friend will appear at $v$ along some edge $e_1$, and then live along some different edge $e_2$. We'll wait some more, and they'll reappear coming from new edge $e_3$, and then leave again along some edge $e_4$.

This will continue until they have arrived or left by every edge that hits $v$. But every time they visit $v$, they must arrive by one edge, and leave by another one, and hence every visit uses up an even number of edges, and so the degree $d(v)$ of $v$ must be even. But there was nothing special about the vertex $v$, and hence the degree of every vertex must be even.

To argue the other way is more difficult; before trying to show there's a closed path that uses all the edges, let's just construct *any* closed path. We pick some vertex $v_0$ to start at, and just randomly choose an edge out of $v_0$, to some other vertex $v_1$, and from there randomly choosing any edge we haven't used yet to another vertex $v_n$, and so on.

To construct a closed walk, we'd like to show we eventually have to return to $v_0$. We're only working with finite graphs, so our walk can't continue forever; the only possibility we have to rule out is that we reach some vertex $v_n$ and find that we have already used every vertex incident to $v_n$. But as we saw before, the path will pair up the edges incident to each vertex as an arriving edge and a departing edge, and we know the degree of $v_n$ is even. If the path has already visited $v_n k$ times, then we'll have used $2k$ of the edges incident to it; when we arrive for the $k+1$st time we'll use one edge, and in all we'll have used $2k+1$ edges, an odd number; since the degree of $v_n$ is even there must be at least one edge we haven't used to exit by.

You might worry that the argument above suggests we can carry on the walk forever, which we obviously can't do since the graph is finite, but the argument above doesn't work for $v_0$: when we start the path out at $v_0$, we haven't had to arrive there, and so the edge we initially leave by is not paired with anything. Therefore, if $\Gamma$ has all vertices with even degree, and we walk randomly for as long as we can, we'll always get stuck at our starting vertex.

Hence, we have shown that if $\Gamma$ is a graph with all vertices having even degree, there will exist some closed walk in $\Gamma$, but the walk we created was chosen randomly, and there's no guarantee it will include all the edge of $\Gamma$ -- in all likelihood, it won't.

But if we look at the edges we haven't used, they will form a simpler graph. $\Gamma'$. There's no reason for $\Gamma'$ to be connected, but it's not too hard to see that every vertex of $\Gamma'$ will still have even degree: in $\Gamma$ every vertex had even degree, and we saw in our first proof that a closed walk that doesn't repeat edges uses up an even number of edges at each vertex, and so we'll have an even number of edges left at each point. Thus, each connected piece of $\Gamma'$ satisfies the hypothesis of the problem, and is simpler, so we can try to find a closed walk on each of connected piece of $\Gamma'$, and then "stitch" the results together to get a walk that uses all the pieces. In the formal proof, this process is best captured using induction, and we'll save the complete description until then, but for now we illustrate the process in an example

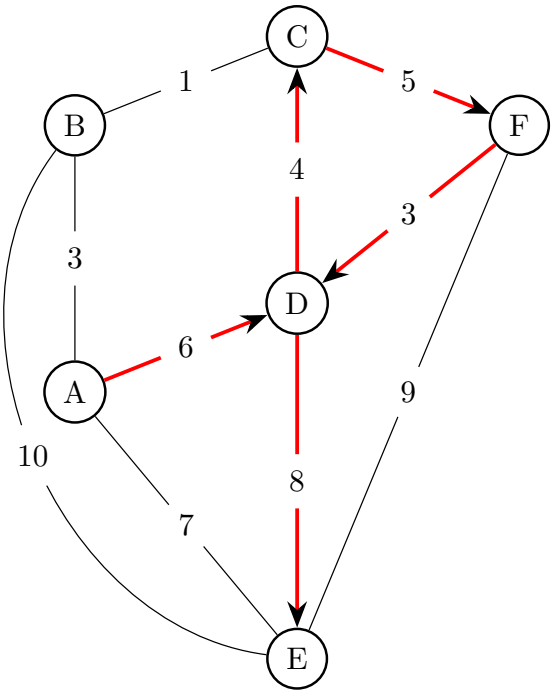



**Figure 2.2.5** Example of a walk

**Example 2.2.6** Let's see an example of how the process of finding an Eulerian path works for the graph Γ in Figure below.

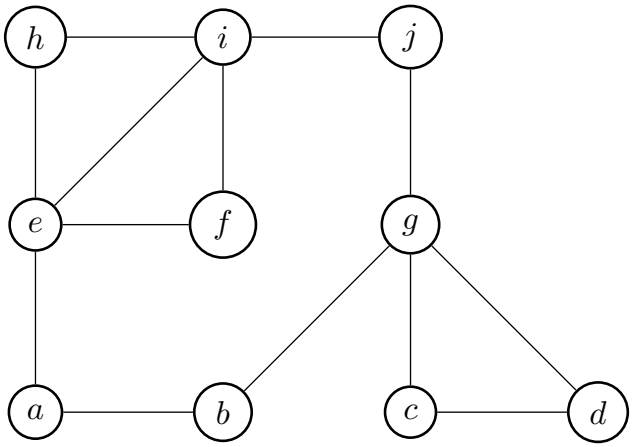



**Figure 2.2.7** A graph Γ

It probably isn't hard to immediately find an Eulerian cycle for Γ just by examination, but to illustrate the algorithm to begin with, we are going to deliberately choose a cycle that doesn't use every edge, the cycle aeijgba show in the next figure:

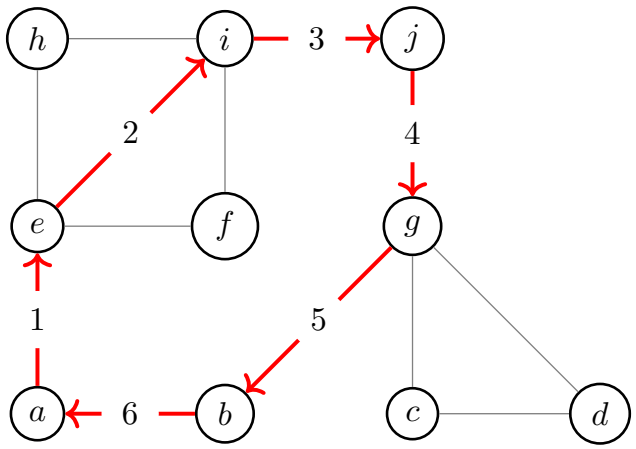



**Figure 2.2.8** Initializing with a closed walk in Γ

To extend our cycle to an Eulerian cycle, we delete all the edges used in the graph, and study the remaining graph. In our case, this graph has two connected components, a four cycle and a three cycle. Both of these are cycles themselves, they're trivially Eulerian, -- in general, it might take some work to find an Eulerian cycle for the components, but won't be too hard as the graph will be smaller.

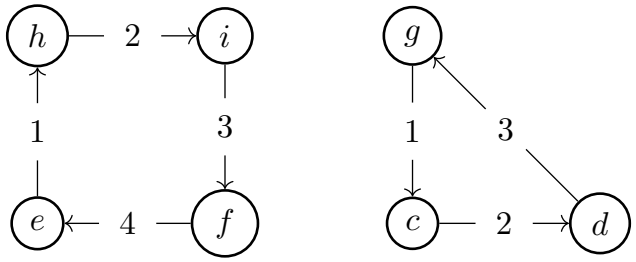



**Figure 2.2.9** Parts of Γ missed by our walk

Finally, we stitch our Eulerian paths together. We follow our initial cycle that wasn't an Eulerian cycle, and the first time we hit a vertex that's in one of the other cycles, we insert that cycle in before we continue along our original path. In our example, our original cycle was aeijgba. a is not eitehr of the other components, but e is in the four cycle, so before we continue on our original cycle to i, we insert the four cycle ehije, giving aehije. We now continue along our original cycle, adding ei, ij, jg, until we reach a vertex g that's in one of the added cycles, which we then insert, giving aehijeijgcdg as our cycle so far.

Continuing this process, we find aehijeijgcdgba as an Eulerian cycle.

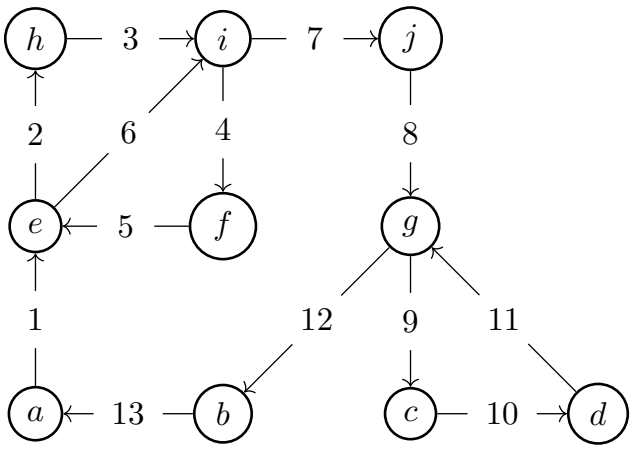



**Figure 2.2.10** The Eulerian cycle stitched together

□

*Proof.* We first show that if $\mathbf{G}$ is Eulerian, then every vertex $v \in \mathbf{G}$ has even degree. For suppose the Eulerian cycle visits the vertex $vk$ times. Each time it visits $v$ it must arrive by one edge, and leave by a different edge. Since the walk is Eulerian, every edge adjacent to $v$ will be used exactly once by the walk, and so we see that $d(v) = 2k$ as desired.

Now we suppose that $\mathbf{G}$ is connected and that every vertex has even degree. We will induct on the number of edges of $\mathbf{G}$. If it has no edges, then the theorem is vacuously true -- we can just take the empty walk.

For the inductive step, suppose that $\mathbf{G}$ is connected with $m$ edges, and that every vertex of $\mathbf{G}$ has even degree. Further assume, for the inductive hypothesis, that every graph $\mathbf{H}$ with these properties and *less* than $m$ edges is Eulerian.

Suppose for now that we can find *a* closed walk $w$ in $\mathbf{G}$ that doesn't repeat any edges -- we will justify that this at the end of the proof. If the closed walk $w$ uses all the edges of $\mathbf{G}$, then $w$ is an Eulerian cycle, and we are done.

If $w$ doesn't use all the edges of $\mathbf{G}$, we can delete all the edges used in $w$ and get a graph $\mathbf{G} \setminus w$ with fewer than $m$ edges. Though $\mathbf{G} \setminus w$ might not be connected, every vertex in $\mathbf{G} \setminus w$ will have even degree, as we've subtracted an even number of edges from each vertex that $w$ visits as argued in the first part of the proof. Thus, by the inductive hypothesis each connected component $\mathbf{G} \setminus w$ will have an Eulerian cycle. Since the $\mathbf{G}$ is connected, $w$ must include at least one point from each component of $\mathbf{G} \setminus w$, and so we can insert the Eulerian walk on the edges of each component of $\mathbf{G} \setminus w$ from each component into $w$ when we reach that component, to obtain a closed walk $w'$ that uses all the edges of $\mathbf{G}$ exactly once, as desired.

All that remains is to justify that $\mathbf{G}$ contains at least one closed walk, given that every vertex has even degree and it contains at least one edge. We form a walk $w$ by starting at any vertex $v$ and at each step choosing any edge we've never traversed before at random as the next step of the walk. We claim that $w$ must eventually return back to the starting vertex $v$. Since $\mathbf{G}$ is finite, and we don't repeat edges, the only way we could fail to return to $v$ would be if our walk "got stuck" -- that is, at some point we reach a vertex $u$ and find that every edge out of $u$ has already been traversed. However, we know that $u$ has an even number of edges, and also arguing as in the first paragraph of the proof that every time the walk $w$ visits $u$ it must use up two edges, one for arrival and one for departure. Hence, when we arrive at $u$ at any time we must have used up an odd number of edges at $u$ -- an even number from all the previous times we have visited $u$, plus one more that we just arrived from. Since $u$ has

even degree, there must always be at least one edge available for us to choose from, and so we can never "get stuck" and will eventually reach $w$ again. ■

**Remark 2.2.11** Note that it does *not* say: "A graph $\Gamma$ is Eulerian if and only if it is connected and every vertex has even degree." This statement in quotation marks is false, but for "stupid" reasons. If $\Gamma$ is Eulerian, and $E_n$ is the graph with $n$ vertices wit no edges, then $\Gamma \sqcup E_n$ is Eulerian but not connected. These are the only examples of such graphs.

**Theorem 2.2.12** *A connected graph $\Gamma$ is semi-Eulerian if and only if it has exactly two vertices with odd degree.*

*Proof.* A minor modification of our argument for Eulerian graphs shows that the condition is necessary. Suppose that $\Gamma$ is semi-Eulerian, with Eulerian path $v_0, e_1, v_1, e_2, v_3, \ldots, e_n, v_n$. Then at any vertex other than the starting or ending vertices, we can pair the entering and leaving edges up to get an even number of edges.

However, at the first vertex $v_0$ the path leaves along $e_1$ the first time but never enters it accordingly, so that $v_0$ has an odd degree; similarly, at $v_n$ the path enters one final time along $e_n$ without leaving, and so $v_n$ also has an odd degree.

To see the condition is sufficient we could also modify the argument for the Eulerian case slightly, but it is slicker instead to *reduce* to the Eulerian case. Suppose that $\Gamma$ is connected, and that vertices $v$ and $w$ have odd degree and all other vertices of $\Gamma$ have even degree. Then we can construct a new graph $\Gamma'$ by adding an extra edge $e = vw$ to $\Gamma$. Then $\Gamma'$ is connected and every vertex has even degree, and so it has an Eulerian cycle. Deleting the edge $e$ that we added from this cycle gives an Eulerian path from $v$ to $w$ in $\Gamma$. ■

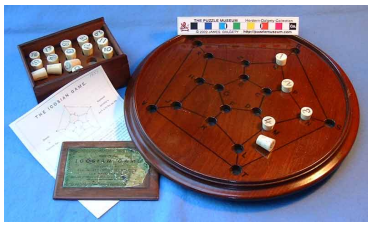
## 2.3 Hamiltonian cycles

We now introduce the concept of Hamiltonian walks. Though on the surface the question seems very similar to determining whether or not a graph is Eulerian, it turns out to be much more difficult.

**Definition 2.3.1** A graph is *Hamiltonian* if it has a closed walk that uses every vertex exactly once; such a path is called a *Hamiltonian cycle* ◊

First, some very basic examples:

1. The cycle graph $C_n$ is Hamiltonian.
2. Any graph obtained from $C_n$ by adding edges is Hamiltonian
3. The path graph $P_n$ is *not* Hamiltonian.



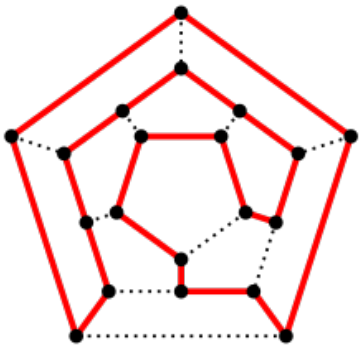


**Figure 2.3.2** The Icosian game (from Puzzle Musuem) and its solution (from Wikipedia)

The term Hamiltonian comes from William Hamiltonian, who invented (a not very successful) board game he termed the "icosian game", which was about finding Hamiltonian cycles on the dodecahedron graph (and possibly its subgraphs).

The main thing you'll need to be able to do with Hamiltonian graphs is decide whether a given graph is Hamiltonian or not. Although the definition of Hamiltonian graph is very similar to that of Eulerian graph, it turns out the two concepts behave very differently. While Euler's Theorem gave us a very easy criterion to check to see whether or not a graph Eulerian, there is no such criterion to see if a graph is Hamiltonian or not. It turns out that deciding whether or not a graph is Hamiltonian is NP-complete, meaning that if we could solve that problem efficiently, then you could solve a host of other difficult problems efficiently as well.

It may seem unfair, then, to ask whether a graph is Hamiltonian or not. But it's only in a very theoretical way that the problem is extremely difficult -- as the number of vertices get very large, the problem gets harder and harderquickly. For any given graph with a low number of vertices, there aren't that many possibilities to check.

If a graph *is* Hamiltonian, then by far the best way to show it is to exhibit a Hamiltonian cycle, as in Figure 2.3.2. When the graph isn't Hamiltonian, things become more interesting.

The most natural way to prove a graph isn't Hamiltonian is to do a case by case analysis of possible paths, showing it doesn't work. For instance, in lecture we outlined the proof that if you remove a vertex from the Icosian graph, than the result isn't Hamiltonian. A natural way to do this is to pick a vertex, and consider the possible pairs of edges the path might take through that vertex. For each possibility, we know some edges won't be used, and can go further along that way.

In general, brute-force case-by-case analyses are proofs we want to avoid when possible, because it can be difficult to make sure we have actually found all the cases, and the proofs aren't always enlightening. It's much better when we can find a "reason" why the graph isn't Hamiltonian.

**Example 2.3.3**

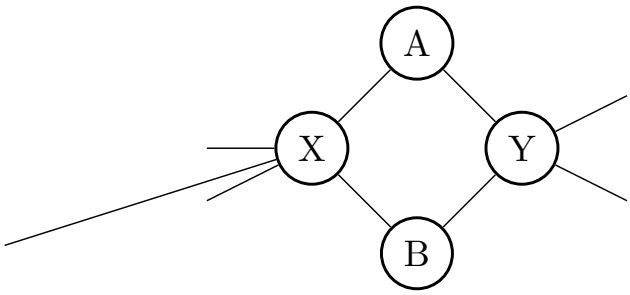



**Figure 2.3.4** A local configuration that can't exist in a Hamiltonian graph

Figure 2.3.4 shows a portion of a larger graph **G**. The exact number of other vertices in the graph that $X$ and $Y$ are adjacent to is not important; what matters is that $A$ and $B$ are each adjacent to two vertices, $X$ and $Y$. Any path through $A$ would have to use $X$ and $Y$, but so would any path through $B$. But then we have a small four cycle $XAYBX$ which doesn't use any other vertices in the graph, and so **G** cannot be HAmiltonian. □

**Lemma 2.3.5** *Suppose that* **G** *is bipartite and Hamiltonian, with* $n$ *red vertices and* $m$ *blue vertices.*

*Proof.* Consider a Hamiltonian path in **G**. Since every edge is between a red and blue vertex, the vertices in the path must alternate between red and blue. Considering every other edge of the cycle pairs each red vertex with a blue vertex, and hence $n = m$. ■

The contrapositive of Lemma 2.3.5 can be used to show graphs *aren't* Hamiltonian: if **G** is bipartite but doesn't have the same number of red vertices and blue vertices, then it can't be Hamiltonian.

**Lemma 2.3.6** *The Petersen Graph is not Hamiltonian*
*Proof.* Of course, a case-by-case analysis of possibile Hamiltonian cycles is possible. The number of cases can be reduced by using symmetries of the Petersen graph. Instead, for variation and to illustrate a different proof technique, we will use another method.

Assume for contradiction that the Petersen graph is Hamiltonian, and draw the ten vertices $v_1, v_2, \ldots, v_{10}$ around the cycle. The Hamiltonian cycle uses 10 of the 15 edges in the Petersen graph, and so there must be 5 more edges, with each vertex incident to one of them, as in the Petersen graph every vertex has degree 3.

Let's analyse where else the edge adjacent to $v_1$ could go. It can't go to $v_1$ itself, as the Petersen graph has no loops, and it can't go to $v_2$ or $v_{10}$ as the Petersen graph has no multiple edges. If it went to $v_3$ or $v_9$ it would make a three cycle, which the Petersen graph doesn't have, and if it went to $v_4$ or $v_7$, there'd be a four cycle. Hence, the only options are $v_1$ is adjacent to $v_5, v_6$ or $v_7$. By reversing the direction of the Hamiltonian path, $v_5$ and $v_7$ are equivalent, and there are only two cases.

But the same analysis holds for every vertex: the extra edge at any vertex can either go to the opposite side of the circle, or be "off by one" and skip three vertices to either direction.

We now claim that not all the extra edges can go "directly across" -- there must be at least one edge that's off by one. If all the extra edges went directly across, then $v_1$ would go to $v_6$, and $v_2$ would go to $v_7$, and $v_1 - v_6 - v_7 - v_2 - v_1$ would be a 4 cycle.

Hence, without loss of generality we may assume that the extra edge at $v_1$ is $v_1 - v_5$. Let us then consider the extra edge at $v_6$. It can't go directly across, as that is $v_1$ which already has its extra edge. Hence it must be off by one, and go to either $v_2$ or $v_{10}$. But either way we get a four cycle: either $v_1 - v_2 - v_6 - v_5 - v_1$, or $v_1 - v_{10} - v_6 - v_5 - v_1$. Hence, we have a contradiction, and the Petersen graph cannot be Hamiltonian. ■

Finally, Ore's Theorem, a positive result, giving conditions which guarantee that a graph has a Hamiltonian cycle. First, a little bit of intuition. If we take an edge to a Hamiltonian graph the result is still Hamiltonian, and the complete graphs $K_n$ are Hamiltonian. Thus, one might expect that a graph with "enough" edges is Hamiltonian.

The trick is in finding an interesting meaning of the word "enough". Simply counting the number of edges does not give very interesting bounds on what "enough" means, however -- the complete graph has $n(n-1)/2$ edges, but we can make it non-Hamiltonian by removing only $n-2$ edges: simply pick a vertex $v$ and remove all but one of the $n-1$ edges coming out of $v$; then $v$ will now have degree 1, and hence the resulting graph cannot be Hamiltonian.

**Theorem 2.3.7 Ore's Theorem.** *Let* **G** *be a simple graph with $n$ vertices, and assume that whenever two distinct vertices $v, w$ are not adjacent, we have $d(v) + d(w) \geq n$. Then* **G** *is Hamiltonian.*
*Proof.* We will argue by contradiction, and begin by passing to a maximal counterexample. Note that if **G** satisfies the hypotheses, and we add an edge

$e$ to $\mathbf{G}$ between two non-adjacent vertices $v$ and $w$, then the result will still satisfy the hypothesis. Indeed, we've only increased the degree of some vertices. So, we had a counterexample $\mathbf{G}$ to Ore's Theorem, we could iteratively add edges to $\mathbf{G}$ that didn't create Hamiltonian cycles, until we got a graph $\mathbf{G}$ that satisfies the hypotheses of Ore's theorem, desn't have any Hamiltonian cycles, but if we add any edge $e$ to $\mathbf{G}$ the result *is* Hamiltonian.

We now observe that such a $\mathbf{G}$ must have a Hamiltonian *path*: indeed, pick any edge $e$ not in $\mathbf{G}$ and add it to $\mathbf{G}$. THe resulting graph is by assumption Hamiltonian, and since $\mathbf{G}$ wasn't Hamiltonian, the Hamiltonian cycle must contain the edge $e$. Deleting the edge $e$ from the Hamiltonian cycle gives a Hamiltonian path in $\mathbf{G}$.

Thus, let $v_1 - v_2 - \cdots v_n$ be a Hamiltonian path in $\mathbf{G}$. We know $v_1$ and $v_n$ are not adjacent, as otherwise $\mathbf{G}$ would be Hamiltonian. Thus, since $\mathbf{G}$ satisfies the hypothesis of Ore's theorem, we know $d(v_1) + d(v_n) \geq n$. We already have one edge adjacent to $v_1$, and one edge adjacent to $v_n$, and so there must be at least $n-2$ other edges adjacent to one or other of these vertices. We will see that no matter how we add $n-2$ edges to these two vertices, we will create a Hamiltonian cycle.

To see this, note there is ever an $i$ with $v_1$ adjacent to $v_i$ and $v_{i-1}$ adjacent to $v_n$, then $\mathbf{G}$ would have a Hamiltonian cycle: namely $v_1 - v_2 - \cdots - v_{i-1} - v_n - v_{n-1} - \vdots v_i - v_1$. Now, there are $n-3$ different vertices we can add edges to $v_1$ to, namely $v_3 - v_{n-1}$, and similarly there are $n-3$ vertices we can add edges connecting $v_n$ to, namely $v_2, \ldots v_{n-2}$. We arrange these $2(n-3)$ edges into a a grid with 2 rows and $n-3$ columns, so that the two edges in each column are $v_1 - v_i$ and $v_n - v_{i-1}$, a pair of edges that can form a Hamiltonian cycle as in the last paragraph.

As we need to add at least $n-2$ edges, but we only have $n-3$ columns, there must be at least one column that contains two edges by the pigeonhole principle, but then we can create a Hamiltonian cycle using those two edges and the edges in our Hamiltonian path. ■

Note that Ore's Theorem is *not* an if and only if, and so Ore's Theorem cannot be used to prove that graphs *aren't* Hamiltonian. Indeed, there are plenty of graphs that are Hamiltonian but do not satisfy the hypotheses of Ore's Theorem. For instance, the cycle graph $C_n$ is Hamiltonian, but every vertex has degree 2, so if $n \geq 5$ the hypotheses of Ore's Theorem are not satisfied.

We also highlight that the proof began by considering a maximal counterexample to Ore's Theorem, and considering maximal/minimal counterexamples is often a useful proof technique, as you the maximality/minimality gives you some extra structure to work with.

## 2.4 Exercises

**1.** The questions in this exercise pertain to the graph $\mathbf{G}$ shown in Figure 2.4.1.

(a) What is the degree of vertex 8?

(b) What is the degree of vertex 10?

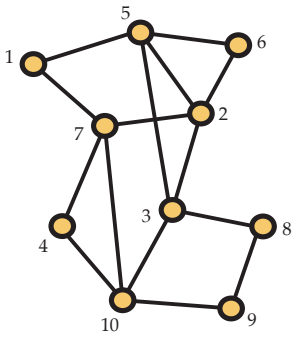
(c) How many vertices of degree 2 are there in **G**? List them.

(d) Find a cycle of length 8 in **G**.

(e) What is the length of a shortest path from 3 to 4?

(f) What is the length of a shortest path from 8 to 7?

(g) Find a path of length 5 from vertex 4 to vertex 6.




**G**

**Figure 2.4.1** A graph

**2.** Draw a graph with 8 vertices, all of odd degree, that does not contain a path of length 3 or explain why such a graph does not exist.

**3.** Find an eulerian circuit in the graph **G** in Figure 2.4.2 or explain why one does not exist.

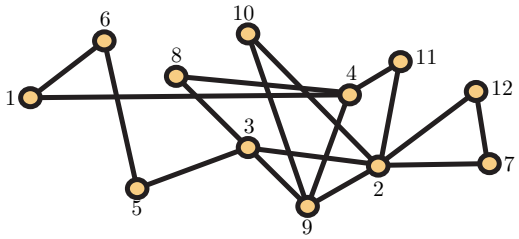



**Figure 2.4.2** A graph **G**

**4.** Consider the graph **G** in Figure 2.4.3. Determine if the graph is eulerian. If it is, find an eulerian circuit. If it is not, explain why it is not. Determine if the graph is hamiltonian. If it is, find a hamiltonian cycle. If it is not, explain why it is not.

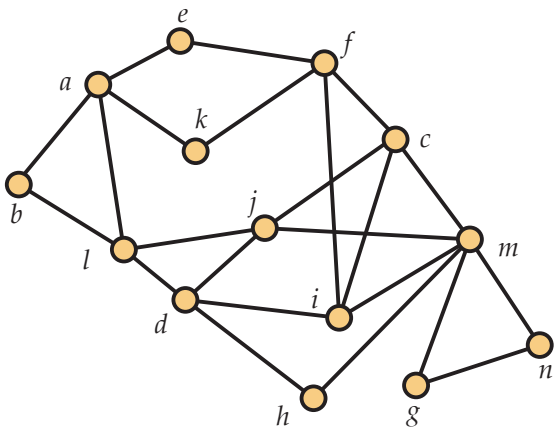



**Figure 2.4.3** A graph **G**

# Chapter 3

# Algorithms

This chapter covers several graph algorithms. We start with two algorithms for finding minimal weight spanning trees, Kruskal's algorithm and Prim's algorithm. We discuss Dijkstra's algorithm for finding the shortest path between two points in a directed, weighted graph.

Much of the material in this chapter is taken from the open source textbook Applied Combinatorics by Keller and Trotter.

## 3.1 Prüfer Codes

This section covers the Prüfer Code, a bijection between labelled trees and certain sequences of integers. This bijection allows us to prove Cayley's theorem, giving a count of such labelled trees.

Given a combinatorial structure, such as a graph or a tree, it is natural to ask how many of such structures there are. Often, there is no nice formula, for instance, for the number of different trees on $n$ vertices there. But if the vertices are labelled, then it turns out there's a nice answer.

**Definition 3.1.1 Labelled tree.** A *labelled tree* on $n$ vertices is a tree with $n$ vertices, which are labelled $1, 2, \ldots, n$. $\Diamond$

**Theorem 3.1.2 Cayley's Theorem.** *There are $n^{n-2}$ labelled trees on $n$ vertices.*

One more convenient way of writing down a labelled tree is to write down all the edges. If there tree has $n$ vertices, then there are $n-1$ edges, and writing down all the edges takes $2n-2$ numbers between $1...n$. However, we see that we're writing down the same tree lots of different times, by changing the order of the edges, and which vertex from each edge we write first. Furthermore, not every sequence of $2n-2$ numbers between $1...n$ will result in a tree.

To fix this problem, we will write down the edges in a particular order. Every tree has at least two leaves, and deleting a leaf gives a small tree. We will use these facts to give a systematic ordering to the edges in a labelled tree, as follows: the first edge will be the edge connecting the leaf with the smallest label to the rest of the tree. We will record that edge, with the leaf on the bottom row, and the "parent" vertex, i.e., the vertex the leaf is connected to, in the top row. Deleting the leaf and the vertex gives a tree with one fewer vertex, and we iterate the process.

**Algorithm 3.1.3 Pruning Algorithm.** *Input: A labelled tree $T$ on $n$ vertices.*

*Output: A $2 \times n-1$ table with entries in $\{1, \ldots, n\}$ that records the edges of $T$ in a specified order.Find the leaf $v$ with the lowest label; it will have one edge $e$, connecting it to some vertex (its "parent") $w$. Form a new tree $T'$ by deleting $v$ and $e$, and record $e$ in the output table, putting the deleted vertex $v$ in the bottom row and its parent $w$ above it in the top row.*

This method fixes the problem of the ordering of the edges not being unique, but as of now we are still recording more information than needed. But note the following: since we delete a vertex when we put it in the bottom row, no number will appear twice on the bottom row. The last column is the last two vertices existing, and if we look at the bottom row and the last entry on the top row, we see that every number from 1 to $n$ will appear exactly once in these spots.

**Definition 3.1.4 Prüfer code.** If record the edges of a tree $T$ as in the Pruning Algorithm, the first $n-2$ number appear in the top row is the *Prüfer code of $T$* ♢

To finish the proof of Cayley's Theorem, we need to show that the Prüfer code is a bijection. The easiest way to do this is to show that it has an inverse; that is, given any sequence of $n-2$ numbers between 1 and $n$, we can construct a tree $T$ have that sequence as its Prüfer code.

This is most easily done by filling in the $n$ numbers we deleted from the table of edges to get the Prüfer code. We will in the numbers on the bottom row from left to right. The first number on the bottom row will be the lowest number that does not appear in the Prüfer code. Delete the first column, and then iterate -- the next number will be the lowest number we haven't used, and that doesn't appear in the remainder of the Prüfer code.

Another way to phrase the last line, is that the next number filled in is always the lowest number the doesn't appear as the bottom entry on one of the $n-1$ columns.

**Example 3.1.5** Suppose $T$ has Prüfer code 4,4,1,4,5,5. This code has length 6, so we looking to complete it by filling in numbers from 1 to 8. We illustrate the process step by step.

The lowest number that doesn't appear is 2, so we fill that in on the bottom of the first column. We no longer have to consider the 4 directly above this 2, as it is not the bottom element of its column.

**Table 3.1.6**

| 4 | 4 | 1 | 4 | 5 | 5 | |
|---|---|---|---|---|---|---|
| 2 | | | | | | |

To fill in the next cell, we put the lowest number not occuring as the lowest element of a column, namely 3.

**Table 3.1.7**

| 4 | 4 | 1 | 4 | 5 | 5 | |
|---|---|---|---|---|---|---|
| 2 | 3 | | | | | |

And now the lowest term not on the bottom of its column is 6, so we add that:

**Table 3.1.8**

| 4 | 4 | 1 | 4 | 5 | 5 | |
|---|---|---|---|---|---|---|
| 2 | 3 | 6 | | | | |

Now the only 1 appearing has an element beneath it, and so 1 gets added in the next column:

**Table 3.1.9**

| 4 | 4 | 1 | 4 | 5 | 5 | |
|---|---|---|---|---|---|---|
| 2 | 3 | 6 | 1 | | | |

And now all the 4s have been passed, so the next number is 4. We jump ahead and fill in the two numbers under 5 as well:

**Table 3.1.10**

| 4 | 4 | 1 | 4 | 5 | 5 | |
|---|---|---|---|---|---|---|
| 2 | 3 | 6 | 1 | 4 | 7 | |

The two numbers we haven't used yet are 5 and 8, so they are the entries in the last column, giving us the completed table of edges

**Table 3.1.11**

| 4 | 4 | 1 | 4 | 5 | 5 | 8 |
|---|---|---|---|---|---|---|
| 2 | 3 | 6 | 1 | 4 | 7 | 5 |

Having constructed the table encoding all the edges, we can now draw the labelled tree with those edges

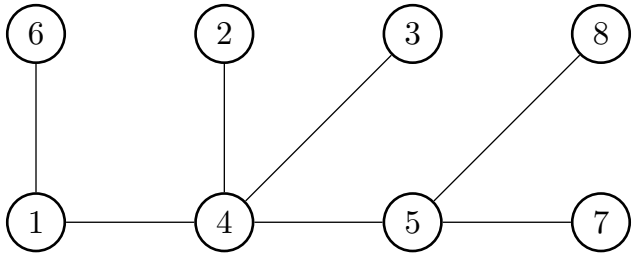



**Figure 3.1.12** The tree with Prüfer code 441455

□

## 3.2 Minimum Weight Spanning Trees

In this section, we consider pairs $(\mathbf{G}, w)$ where $\mathbf{G} = (V, E)$ is a connected graph and $w\colon E \to \mathbb{N}_0$. For each edge $e \in E$, the quantity $w(e)$ is called the **weight** of $e$. Given a set $S$ of edges, we define the **weight** of $S$, denoted $w(S)$, by setting $w(S) = \sum_{e \in S} w(e)$. In particular, the weight of a spanning tree $T$ is just the sum of the weights of the edges in $T$.

Weighted graphs arise in many contexts. One of the most natural is when the weights on the edges are distances or costs. For example, consider the weighted graph in Figure 3.2.1. Suppose the vertices represent nodes of a network and the edges represent the ability to establish direct physical connections between those nodes. The weights associated to the edges represent the cost (let's say in thousands of dollars) of building those connections. The company establishing the network among the nodes only cares that there is a way to get data between each pair of nodes. Any additional links would create redundancy in which they are not interested at this time. A spanning tree of the graph ensures that each node can communicate with each of the others and has no redundancy, since removing any edge disconnects it. Thus, to minimize the cost of building the network, we want to find a minimum weight (or cost) spanning tree.

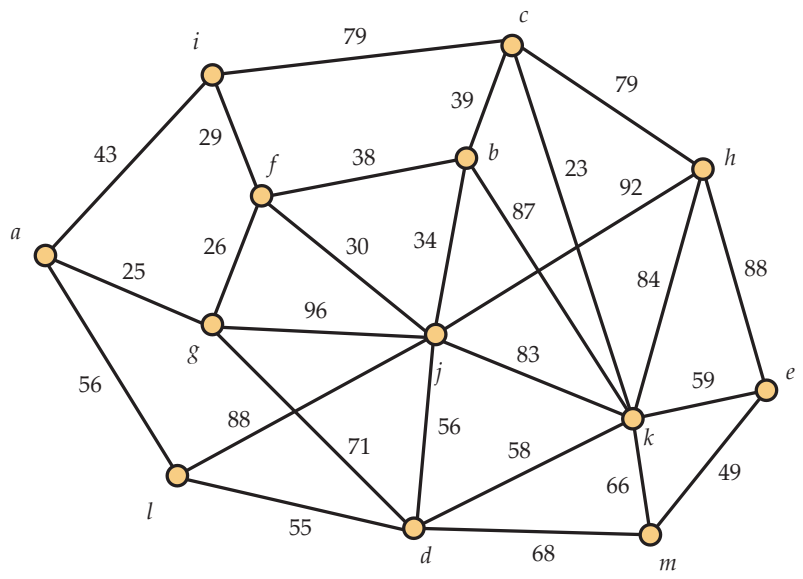



**Figure 3.2.1** A weighted graph

To do this, this section considers the following problem:

**Problem 3.2.2** Find a minimum weight spanning tree $\mathbf{T}$ of $\mathbf{G}$. □

To solve this problem, we will develop *two* efficient graph algorithms, each having certain computational advantages and disadvantages. Before developing the algorithms, we need to establish some preliminaries about spanning trees and forests.

### 3.2.1 Preliminaries

The following proposition about the number of components in a spanning forest of a graph $\mathbf{G}$ has an easy inductive proof. You are asked to provide it in the exercises.

**Proposition 3.2.3** *Let $\mathbf{G} = (V, E)$ be a graph on $n$ vertices, and let $\mathbf{H} = (V, S)$ be a spanning forest. Then $0 \leq |S| \leq n-1$. Furthermore, if $|S| = n-k$, then $\mathbf{H}$ has $k$ components. In particular, $\mathbf{H}$ is a spanning tree if and only if it contains $n-1$ edges.*

The following proposition establishes a way to take a spanning tree of a graph, remove an edge from it, and add an edge of the graph that is not in the spanning tree to create a new spanning tree. Effectively, the process exchanges two edges to form the new spanning tree, so we call this the **exchange principle**.

**Proposition 3.2.4 Exchange Principle.** *Let $\mathbf{T} = (V, S)$ be spanning tree in a graph $\mathbf{G}$, and let $e = xy$ be an edge of $\mathbf{G}$ which does not belong to $\mathbf{T}$. Then*

1. *There is a* unique *path $P = (x_0, x_1, x_2, \ldots, x_t)$ with (a) $x = x_0$; (b) $y = x_t$; and (c) $x_i x_{i+1} \in S$ for each $i = 0, 1, 2, \ldots, t-1$.*

2. *For each $i = 0, 1, 2, \ldots, t-1$, let $f_i = x_i x_{i+1}$ and then set*

$$S_i = \{e\} \cup \{g \in S : g \neq f_i\},$$

*i.e., we **exchange** edge $f_i$ for edge $e$. Then $\mathbf{T}_i = (V, S_i)$ is a spanning tree of $\mathbf{G}$.*

*Proof.* For the first fact, it suffices to note that if there were more than one distinct path from $x$ to $y$ in $\mathbf{T}$, we would be able to find a cycle in $\mathbf{T}$. This is impossible since it is a tree. For the second, we refer to Figure 3.2.5. The black and green edges in the graph shown at the left represent the spanning tree $\mathbf{T}$. Thus, $f$ lies on the unique path from $x$ to $y$ in $\mathbf{T}$ and $e = xy$ is an edge of $\mathbf{G}$ *not* in $\mathbf{T}$. Adding $e$ to $\mathbf{T}$ creates a graph with a unique cycle, since $\mathbf{T}$ had a unique path from $x$ to $y$. Removing $f$ (which could be any edge $f_i$ of the path, as stated in the proposition) destroys this cycle. Thus $\mathbf{T}_i$ is a connected acyclic subgraph of $\mathbf{G}$ with $n - 1 + 1 - 1 = n - 1$ edges, so it is a spanning tree.

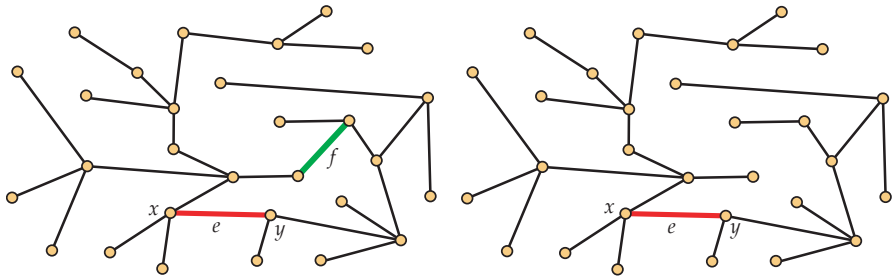



**Figure 3.2.5** The exchange principle

■

For both of the algorithms we develop, the argument to show that the algorithm is optimal rests on the following technical lemma. To avoid trivialities, we assume $n \geq 3$.

**Lemma 3.2.6** *Let* $\mathbf{F}$ *be a spanning forest of* $\mathbf{G}$ *and let* $C$ *be a component of* $\mathbf{F}$*. Also, let* $e = xy$ *be an edge of minimum weight among all edges with one endpoint in* $C$ *and the other not in* $C$*. Then among all spanning trees of* $\mathbf{G}$ *that contain the forest* $\mathbf{F}$*, there is one of minimum weight that contains the edge* $e$*.*

*Proof.* Let $\mathbf{T} = (V, S)$ be any spanning tree of minimum weight among all spanning trees that contain the forest $\mathbf{F}$, and suppose that $e = xy$ is not an edge in $\mathbf{T}$. (If it were an edge in $\mathbf{T}$, we would be done.) Then let $P = (x_0, x_1, x_2, \ldots, x_t)$ be the unique path in $\mathbf{T}$ with (a) $x = x_0$; (b) $y = x_t$; and (c) $x_i x_{i+1} \in S$ for each $i = 0, 1, 2, \ldots, t - 1$. Without loss of generality, we may assume that $x = x_0$ is a vertex in $C$ while $y = x_t$ does not belong to $C$. Then there is a least non-negative integer $i$ for which $x_i$ is in $C$ and $x_{i+1}$ is not in $C$. It follows that $x_j$ is in $C$ for all $j$ with $0 \leq j \leq i$.

Let $f = x_i x_{i+1}$. The edge $e$ has minimum weight among all edges with one endpoint in $C$ and the other not in $C$, so $w(e) \leq w(f)$. Now let $\mathbf{T}_i$ be the tree obtained by exchanging the edge $f$ for edge $e$. It follows that $w(\mathbf{T}_i) = w(\mathbf{T}) - w(f) + w(e) \leq w(\mathbf{T})$. Furthermore, $\mathbf{T}_i$ contains the spanning forest $\mathbf{F}$ as well as the edge $e$. It is therefore the minimum weight spanning tree we seek. ■

**Remark 3.2.7** Although Bob's combinatorial intuition has improved over the course he doesn't quite understand why we need special algorithms to find minimum weight spanning trees. He figures there can't be that many spanning trees, so he wants to just write them down. Alice groans as she senses that Bob must have been absent when the material from Section 3.1 was discussed. In that section, we learned that a graph on $n$ vertices can have as many as $n^{n-2}$ spanning trees (or horrors, the instructor may have left it off the syllabus). Regardless, this exhaustive approach is already unusable when $n = 20$. Dave mumbles something about being greedy and just adding the lightest edges one-by-one while never adding an edge that would make a cycle. Zori remembers a strategy like this working for finding the height of a poset, but she's worried

about the nightmare situation that we learned about with using FirstFit to color graphs. Alice agrees that greedy algorithms have an inconsistent track record but suggests that Lemma 3.2.6 may be enough to get one to succeed here.

### 3.2.2 Kruskal's Algorithm

In this section, we develop one of the best known algorithms for finding a minimum weight spanning tree. It is known as **Kruskal's Algorithm**, although some prefer the descriptive label *Avoid Cycles* because of the way it builds the spanning tree.

To start Kruskal's algorithm, we sort the edges according to weight. To be more precise, let $m$ denote the number of edges in $\mathbf{G} = (V, E)$. Then label the edges as $e_1, e_2, e_3, \ldots, e_m$ so that $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$. Any of the many available efficient sorting algorithms can be used to do this step.

Once the edges are sorted, Kruskal's algorithm proceeds to an initialization step and then inductively builds the spanning tree $\mathbf{T} = (V, S)$:

**Algorithm 3.2.8 Kruskal's Algorithm.**

**Initialization.** *Set $S = \emptyset$ and $i = 0$.*

**Inductive Step.** *While $|S| < n - 1$, let $j$ be the least non-negative integer so that $j > i$ and there are no cycles in $S \cup \{e_j\}$. Then (using pseudo-code) set*

$$i = j \quad \textit{and} \quad S = S \cup \{j\}.$$

The correctness of Kruskal's Algorithm follows from an inductive argument. First, the set $S$ is initialized as the empty set, so there is certainly a minimum weight spanning tree containing all the edges in $S$. Now suppose that for some $i$ with $0 \leq i < n$, $|S| = i$ and there is a minimum weight spanning tree containing all the edges in $S$. Let $\mathbf{F}$ be the spanning forest determined by the edges in $S$, and let $C_1, C_2, \ldots, C_s$ be the components of $\mathbf{F}$. For each $k = 1, 2, \ldots, s$, let $f_k$ be a minimum weight edge with one endpoint in $C_k$ and the other not in $C_k$. Then the edge $e$ added to $S$ by Kruskal's Algorithm is just the edge $\{f_1, f_2, \ldots, f_s\}$ having minimum weight. Applying Lemma 3.2.6 and the inductive hypothesis, we know that there will still be a minimum weight spanning tree of $\mathbf{G}$ containing all the edges of $S \cup \{e\}$.

**Example 3.2.9 Kruskal's Algorithm.**

Let's see what Kruskal's algorithm does on the weighted graph in Figure 3.2.1. It first sorts all of the edges by weight. We won't reproduce the list here, since we won't need all of it. The edge of least weight is $ck$, which has weight 23. It continues adding the edge of least weight, adding $ag$, $fg$, $fi$, $fj$, and $bj$. However, after doing this, the edge of lowest weight is $fb$, which has weight 38. This edge cannot be added, as doing so would make $fjb$ a cycle. Thus, the algorithm bypasses it and adds $bc$. Edge $ai$ is next inspected, but it, too, would create a cycle and is eliminated from consideration. Then $em$ is added, followed by $dl$. There are now *two* edges of weight 56 to be considered: $al$ and $dj$. Our sorting algorithm has somehow decided one of them should appear first, so let's say it's $dj$. After adding $dj$, we cannot add $al$, as $agfjdl$ would form a cycle. Edge $dk$ is next considered, but it would also form a cycle. However, $ek$ can be added. Edges $km$ and $dm$ are then bypassed. Finally, edge $ch$ is added as the twelfth and final edge for this 13-vertex spanning tree. The full list of edges added (in order) is shown to the right. The total weight of this spanning tree is 504.

```
c k 23
a g 25
f g 26
f i 29
f j 30
b j 34
b c 39
e m 49
d l 55
d j 56
e k 59
c h 79
```

□

### 3.2.3 Prim's Algorithm

We now develop **Prim's Algorithm** for finding a minimum weight spanning tree. This algorithm is also known by a more descriptive label: *Build Tree*. We begin by choosing a root vertex $r$. Again, the algorithm proceeds with an initialization step followed by a series of inductive steps.

**Algorithm 3.2.10 Prim's Algorithm.**

**Initialization.** *Set* $W = \{r\}$ *and* $S = \emptyset$.

**Inductive Step.** *While* $|W| < n$, *let* $e$ *be an edge of minimum weight among all edges with one endpoint in* $W$ *and the other not in* $W$. *If* $e = xy$, $x \in W$ *and* $y \notin W$, *update* $W$ *and* $S$ *by setting (using pseudo-code)*

$$W = W \cup \{y\} \quad \textit{and} \quad S = S \cup \{e\}.$$

The correctness of Prim's algorithm follows immediately from Lemma 3.2.6.

**Example 3.2.11 Prim's Algorithm.**

Let's see what Prim's algorithm does on the weighted graph in Figure 3.2.1. We start with vertex $a$ as the root vertex. The lightest edge connecting $a$ (the only vertex in the tree so far) to the rest of the graph is $ag$. Next, $fg$ is added. This is followed by $fi$, $fj$, $bj$, and $bc$. Next, the algorithm identifies $ck$ as the lightest edge connecting $\{a, g, i, f, j, b, c\}$ to the remaining vertices. Notice that this is considerably later than Kruskal's algorithm finds the same edge. The algorithm then determines that $al$ and $jd$, both of weight 56 are the lightest edges connecting vertices in the tree to the other vertices. It picks arbitrarily, so let's say it takes $al$. It next finds $dl$, then $ek$, and then $em$. The final edge added is $ch$. The full list of edges added (in order) is shown to the right. The total weight of this spanning tree is 504. This (not surprisingly) the same weight we obtained using Kruskal's algorithm. However, notice that the spanning tree found is different, as this one contains $al$ instead of $dj$. This is not an issue, of course, since in both cases an arbitrary choice between two edges of equal weight was made.

```
a g 25
f g 26
f i 29
f j 30
b j 34
b c 39
c k 23
a l 56
d l 55
e k 59
e m 49
c h 79
```

□

### 3.2.4 Comments on Efficiency

An implementation of Kruskal's algorithm seems to require that the edges be sorted. If the graph has $n$ vertices and $m$ edges, this requires $m \log m$ operations just for the sort. But once the sort is done, the process takes only $n - 1$ steps—provided you keep track of the components as the spanning forest expands. Regardless, it is easy to see that at most $O(n^2 \log n)$ operations are required.

On the other hand, an implementation of Prim's algorithm requires the program to conveniently keep track of the edges incident with each vertex and always be able to identify the edge with least weight among subsets of these edges. In computer science, the data structure that enables this task to be carried out is called a **heap**.

## 3.3 Digraphs

In this section, we introduce another useful variant of a graph. In a graph, the existence of an edge $xy$ can be used to model a connection between $x$ and $y$ that goes in both ways. However, sometimes such a model is insufficient. For instance, perhaps it is possible to fly from Atlanta directly to Fargo but not possible to fly from Fargo directly to Atlanta. In a graph representing the airline network, an edge between Atlanta and Fargo would lose the information that the flights only operate in one direction. To deal with this problem, we introduce a new discrete structure. A **digraph G** is a pair $(V, E)$ where $V$ is a vertex set and $E \subset V \times V$ with $x \neq y$ for every $(x, y) \in E$. We consider the pair $(x, y)$ as a **directed edge** from $x$ to $y$. Note that for distinct vertices $x$ and $y$ from $V$, the ordered pairs $(x, y)$ and $(y, x)$ are distinct, so the digraph may have one, both or neither of the directed edges $(x, y)$ and $(y, x)$. This is in contrast to graphs, where edges are sets, so $\{x, y\}$ and $\{y, x\}$ are the same.

Diagrams of digraphs use arrowheads on the edges to indicate direction. This is illustrated in Figure 3.3.1. For example, the digraph illustrated there contains the edge $(a, f)$ but not the edge $(f, a)$. It does contain both edges $(c, d)$ and $(d, c)$, however.

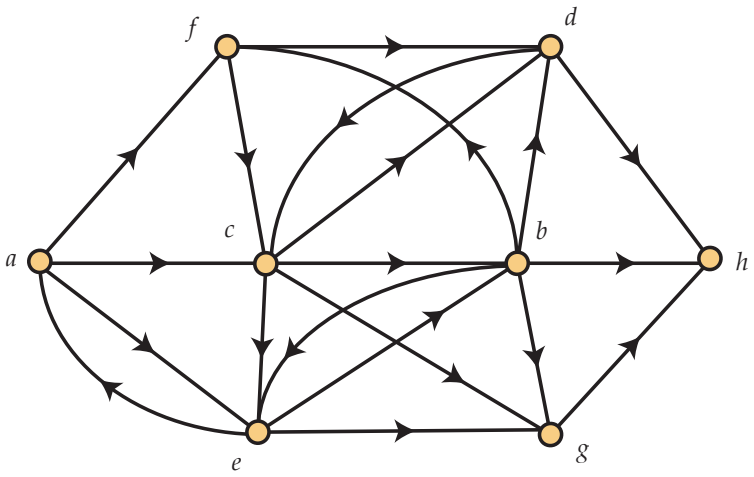



**Figure 3.3.1** A Digraph

When **G** is a digraph, a sequence $P = (r = u_0, u_1, \ldots, u_t = x)$ of distinct vertices is called a **directed path** from $r$ to $x$ when $(u_i u_{i+1})$ is a directed edge in **G** for every $i = 0, 1, \ldots, t-1$. A directed path $C = (r = u_0, u_1, \ldots, u_t = x)$ is called a **directed cycle** when $(u_t, u_0)$ is a directed edge of **G**.

## 3.4 Dijkstra's Algorithm for Shortest Paths

Just as with graphs, it is useful to assign weights to the directed edges of a digraph. Specifically, in this section we consider a pair $(\mathbf{G}, w)$ where $\mathbf{G} = (V, E)$ is a digraph and $w\colon E \to \mathbb{N}_0$ is a function assigning to each directed edge $(x, y)$ a non-negative weight $w(x, y)$. However, in this section, we interpret weight as **distance** so that $w(x, y)$ is now called the **length** of the edge $(x, y)$. If $P = (r = u_0, u_1, \ldots, u_t = x)$ is a directed path from $r$ to $x$, then the **length** of the path $P$ is just the sum of the lengths of the edges in the path, $\sum_{i=0}^{t-1} w(u_i u_{i+1})$. The **distance** from $r$ to $x$ is then defined to be the minimum length of a directed path from $r$ to $x$. Our goal in this section is to solve the following natural problem, which has many applications:

**Problem 3.4.1** For each vertex $x$, find the distance from $r$ to $x$. Also, find a shortest path from $r$ to $x$. □

### 3.4.1 Description of the Algorithm

To describe **Dijkstra's algorithm** in a compact manner, it is useful to extend the definition of the function $w$. We do this by setting $w(x, y) = \infty$ when $x \neq y$ and $(x, y)$ is not a directed edge of **G**. In this way, we will treat $\infty$ as if it were a number (although it is not!).[1]

We are now prepared to describe Dijkstra's Algorithm.

**Algorithm 3.4.2 Dijkstra's Algorithm.** *Let $n = |V|$. At Step $i$, where $1 \leq i \leq n$, we will have determined:*

1. *A sequence $\sigma = (v_1, v_2, v_3, \ldots, v_i)$ of distinct vertices from* **G** *with $r = v_1$. These vertices are called* ***permanent vertices****, while the remaining vertices will be called* ***temporary vertices****.*

[1] This is not an issue for computer implementation of the algorithm, as instead of using $\infty$, a value given by the product of the number of vertices and the maximum edge weight may be used to simulate infinity.

2. *For each vertex $x \in V$, we will have determined a number $\delta(x)$ and a path $P(x)$ from $r$ to $x$ of length $\delta(x)$.*

**Initialization (Step 1)** *Set $i = 1$. Set $\delta(r) = 0$ and let $P(r) = (r)$ be the trivial one-point path. Also, set $\sigma = (r)$. For each $x \neq r$, set $\delta(x) = w(r, x)$ and $P(x) = (r, x)$. Let $x$ be a temporary vertex for which $\delta(x)$ is minimum. Set $v_2 = x$, and update $\sigma$ by appending $v_2$ to the end of it. Increment $i$.*

**Inductive Step (Step $i$, $i > 1$)** *If $i < n$, then for each temporary $x$, let*

$$\delta(x) = \min\{\delta(x), \delta(v_i) + w(v_i, x)\}.$$

*If this assignment results in a reduction in the value of $\delta(x)$, let $P(x)$ be the path obtained by adding $x$ to the end of $P(v_i)$.*

*Let $x$ be a temporary vertex for which $\delta(x)$ is minimum. Set $v_{i+1} = x$, and update $\sigma$ by appending $v_{i+1}$ to it. Increment $i$.*

### 3.4.2 Example of Dijkstra's Algorithm

Before establishing why Dijkstra's algorithm works, it may be helpful to see an example of how it works. To do this, consider the digraph **G** shown in Figure 3.4.3. For visual clarity, we have chosen a digraph which is an **oriented graph**, i.e., for each distinct pair $x, y$ of vertices, the graph contains at most one of the two possible directed edges $(x, y)$ and $(y, x)$.

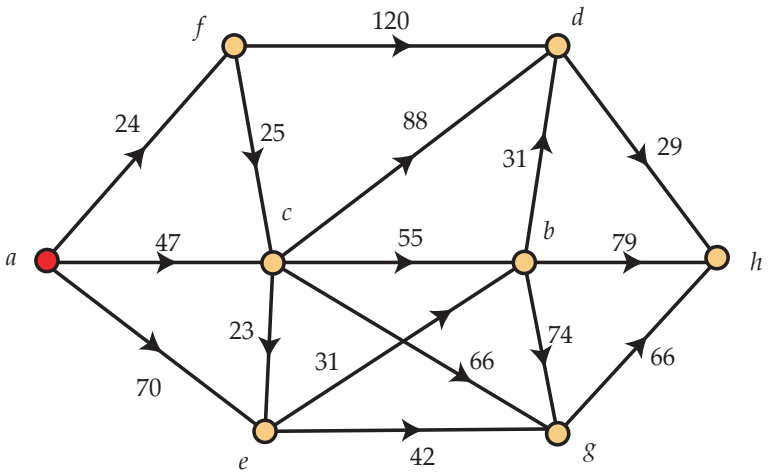



**Figure 3.4.3** A digraph with edge lengths

Suppose that the root vertex $r$ is the vertex labeled $a$. The initialization step of Dijkstra's algorithm then results in the following values for $\delta$ and $P$:

**Step 1. Initialization.**

$$\begin{aligned} \sigma &= (a) \\ \delta(a) &= 0; & P(a) &= (a) \\ \delta(b) &= \infty; & P(b) &= (a, b) \\ \delta(c) &= 47; & P(c) &= (a, c) \\ \delta(d) &= \infty; & P(d) &= (a, d) \\ \delta(e) &= 70; & P(e) &= (a, e) \\ \delta(f) &= 24; & P(f) &= (a, f) \end{aligned}$$

$$\delta(g) = \infty; \qquad P(g) = (a, g)$$
$$\delta(h) = \infty; \qquad P(h) = (a, h)$$

Before finishing Step 1, the algorithm identifies vertex $f$ as closest to $a$ and appends it to $\sigma$, making $a$ permanent. When entering Step 2, Dijkstra's algorithm attempts to find shorter paths from $a$ to each of the temporary vertices by going through $f$. We call this process "scanning from vertex $f$." In this scan, the path to vertex $d$ is updated, since $\delta(f) + w(f, d) = 24 + 120 = 144 < \infty = w(a, d)$.

**Step 2. Scan from vertex $f$.**

$$\sigma = (a, f)$$
$$\delta(a) = 0; \qquad P(a) = (a)$$
$$\delta(b) = \infty; \qquad P(b) = (a, b)$$
$$\delta(c) = 47; \qquad P(c) = (a, c)$$
$$\delta(d) = 144 = 24 + 120 = \delta(f) + w(f, d); \qquad P(d) = (a, f, d) \quad \text{updated}$$
$$\delta(e) = 70; \qquad P(e) = (a, e)$$
$$\delta(f) = 24; \qquad P(f) = (a, f)$$
$$\delta(g) = \infty; \qquad P(g) = (a, f)$$
$$\delta(h) = \infty; \qquad P(h) = (a, h)$$

Before proceeding to the next step, vertex $c$ is made permanent by making it $v_3$. In Step 3, therefore, the scan is from vertex $c$. Vertices $b$, $d$, and $g$ have their paths updated. However, although $\delta(c) + w(c, e) = 47 + 23 = 70 = \delta(e)$, we do not change $P(e)$ since $\delta(e)$ is not *decreased* by routing $P(e)$ through $c$.

**Step 3. Scan from vertex $c$.**

$$\sigma = (a, f, c)$$
$$\delta(a) = 0; \qquad P(a) = (a)$$
$$\delta(b) = 102 = 47 + 55 = \delta(c) + w(c, b); \qquad P(b) = (a, c, b) \quad \text{updated}$$
$$\delta(c) = 47; \qquad P(c) = (a, c)$$
$$\delta(d) = 135 = 47 + 88 = \delta(c) + w(c, d); \qquad P(d) = (a, c, d) \quad \text{updated}$$
$$\delta(e) = 70; \qquad P(e) = (a, e)$$
$$\delta(f) = 24; \qquad P(f) = (a, f)$$
$$\delta(g) = 113 = 47 + 66 = \delta(c) + w(c, g); \qquad P(g) = (a, c, g) \quad \text{updated}$$
$$\delta(h) = \infty; \qquad P(h) = (a, h)$$

Now vertex $e$ is made permanent.

**Step 4. Scan from vertex $e$.**

$$\sigma = (a, f, c, e)$$
$$\delta(a) = 0; \qquad P(a) = (a)$$
$$\delta(b) = 101 = 70 + 31 = \delta(e) + w(e, b); \qquad P(b) = (a, e, b) \quad \text{updated}$$
$$\delta(c) = 47; \qquad P(c) = (a, c)$$
$$\delta(d) = 135; \qquad P(d) = (a, c, d)$$
$$\delta(e) = 70; \qquad P(e) = (a, e)$$

$\delta(f) = 24;$ $P(f) = (a, f)$
$\delta(g) = 112 = 70 + 42 = \delta(e) + w(e, g);$ $P(g) = (a, e, g)$ updated
$\delta(h) = \infty;$ $P(h) = (a, h)$

Now vertex $b$ is made permanent.

**Step 5. Scan from vertex $b$.**

$\sigma = (a, f, c, e, b)$
$\delta(a) = 0;$ $P(a) = (a)$
$\delta(b) = 101;$ $P(b) = (a, e, b)$
$\delta(c) = 47;$ $P(c) = (a, c)$
$\delta(d) = 132 = 101 + 31 = \delta(b) + w(b, d);$ $P(d) = (a, e, b, d)$ updated
$\delta(e) = 70;$ $P(e) = (a, e)$
$\delta(f) = 24;$ $P(f) = (a, f)$
$\delta(g) = 112;$ $P(g) = (a, e, g)$
$\delta(h) = 180 = 101 + 79 = \delta(b) + w(b, h);$ $P(h) = (a, e, b, h)$ updated

Now vertex $g$ is made permanent.

**Step 6. Scan from vertex $g$.**

$\sigma = (a, f, c, e, b, g)$
$\delta(a) = 0;$ $P(a) = (a)$
$\delta(b) = 101;$ $P(b) = (a, e, b)$
$\delta(c) = 47;$ $P(c) = (a, c)$
$\delta(d) = 132;$ $P(d) = (a, e, b, d)$
$\delta(e) = 70;$ $P(e) = (a, e)$
$\delta(f) = 24;$ $P(f) = (a, f)$
$\delta(g) = 112;$ $P(g) = (a, e, g)$
$\delta(h) = 178 = 112 + 66 = \delta(g) + w(g, h);$ $P(h) = (a, e, g, h)$ updated

Now vertex $d$ is made permanent.

**Step 7. Scan from vertex $d$.**

$\sigma = (a, f, c, e, b, g, d)$
$\delta(a) = 0;$ $P(a) = (a)$
$\delta(b) = 101;$ $P(b) = (a, e, b)$
$\delta(c) = 47;$ $P(c) = (a, c)$
$\delta(d) = 132;$ $P(d) = (a, e, b, d)$
$\delta(e) = 70;$ $P(e) = (a, e)$
$\delta(f) = 24;$ $P(f) = (a, f)$
$\delta(g) = 112;$ $P(g) = (a, e, g)$
$\delta(h) = 161 = 132 + 29 = \delta(d) + w(d, h);$ $P(h) = (a, e, b, d, h)$ updated

Now vertex $h$ is made permanent. Since this is the last vertex, the algorithm halts and returns the following:

**Final Results of Dijkstra's Algorithm.**

$$\sigma = (a, f, c, e, b, g, d, h)$$

| | |
|---|---|
| $\delta(a) = 0;$ | $P(a) = (a)$ |
| $\delta(b) = 101;$ | $P(b) = (a, e, b)$ |
| $\delta(c) = 47;$ | $P(c) = (a, c)$ |
| $\delta(d) = 132;$ | $P(d) = (a, e, b, d)$ |
| $\delta(e) = 70;$ | $P(e) = (a, e)$ |
| $\delta(f) = 24;$ | $P(f) = (a, f)$ |
| $\delta(g) = 112;$ | $P(g) = (a, e, g)$ |
| $\delta(h) = 161;$ | $P(h) = (a, e, b, d, h)$ |

### 3.4.3 The Correctness of Dijkstra's Algorithm

Now that we've illustrated Dijkstra's algorithm, it's time to prove that it actually does what we claimed it does: find the distance from the root vertex to each of the other vertices and a path of that length. To do this, we first state two elementary propositions. The first is about shortest paths in general, while the second is specific to the sequence of permanent vertices produced by Dijkstra's algorithm.

**Proposition 3.4.4** *Let $x$ be a vertex and let $P = (r = u_0, u_1, \ldots, u_t = x)$ be a shortest path from $r$ to $x$. Then for every integer $j$ with $0 < j < t$, $(u_0, u_1, \ldots, u_j)$ is a shortest path from $r$ to $u_j$ and $(u_j, u_{j+1}, \ldots, u_t)$ is a shortest path from $u_j$ to $u_t$*

**Proposition 3.4.5** *When the algorithm halts, let $\sigma = (v_1, v_2, v_3, \ldots, v_n)$. Then*

$$\delta(v_1) \leq \delta(v_2) \leq \cdots \leq \delta(v_n).$$

We are now ready to prove the correctness of the algorithm. The proof we give will be inductive, but the induction will have nothing to do with the total number of vertices in the digraph or the step number the algorithm is in.

**Theorem 3.4.6** *Dijkstra's algorithm yields shortest paths for every vertex $x$ in $\mathbf{G}$. That is, when Dijkstra's algorithm terminates, for each $x \in V$, the value $\delta(x)$ is the distance from $r$ to $x$ and $P(x)$ is a shortest path from $r$ to $x$.*

*Proof.* The theorem holds trivially when $x = r$. So we consider the case where $x \neq r$. We argue that $\delta(x)$ is the distance from $r$ to $x$ and that $P(x)$ is a shortest path from $r$ to $x$ by induction on the minimum number $k$ of edges in a shortest path from $r$ to $x$. When $k = 1$, the edge $(r, x)$ is a shortest path from $r$ to $x$. Since $v_1 = r$, we will set $\delta(x) = w(r, x)$ and $P(x) = (r, x)$ at Step 1.

Now fix a positive integer $k$. Assume that if the minimum number of edges in a shortest path from $r$ to $x$ is at most $k$, then $\delta(x)$ is the distance from $r$ to $x$ and $P(x)$ is a shortest path from $r$ to $x$. Let $x$ be a vertex for which the minimum number of edges in a shortest path from $r$ to $x$ is $k + 1$. Fix a shortest path $P = (u_0, u_1, u_2, \ldots, u_{k+1})$ from $r = u_0$ to $x = u_{k+1}$. Then $Q = (u_0, u_1, \ldots, u_k)$ is a shortest path from $r$ to $u_k$. (See Figure 3.4.7.)

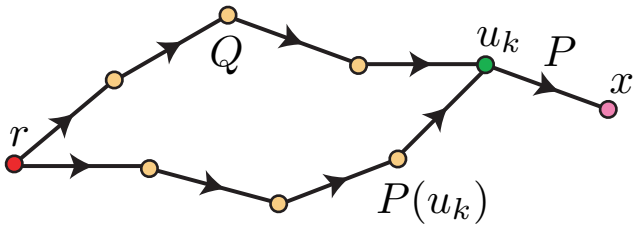



**Figure 3.4.7** Shortest paths

By the inductive hypothesis, $\delta(u_k)$ is the distance from $r$ to $u_k$, and $P(u_k)$ is a shortest path from $r$ to $u_k$. Note that $P(u_k)$ need not be the same as path $Q$, as we suggest in Figure 3.4.7. However, if distinct, the two paths will have the same length, namely $\delta(u_k)$. Also, the distance from $r$ to $x$ is $\delta(u_k)+w(u_k,x) \geq \delta(u_k)$ since $P$ is a shortest path from $r$ to $x$ and $w(u_k,x) \geq 0$.

Let $i$ and $j$ be the unique integers for which $u_k = v_i$ and $x = v_j$. If $j < i$, then

$$\delta(x) = \delta(v_j) \leq \delta(v_i) = \delta(u_k) \leq \delta(u_k) + w(u_k).$$

Therefore the algorithm has found a path $P(x)$ from $r$ to $x$ having length $\delta(x)$ which is at most the distance from $r$ to $x$. Clearly, this implies that $\delta(x)$ is the distance from $r$ to $x$ and that $P(x)$ is a shortest path.

On the other hand, if $j > i$, then the inductive step at Step $i$ results in

$$\delta(x) \leq \delta(v_i) + w(v_i, y) = \delta(u_k) + w(u_k, x).$$

As before, this implies that $\delta(x)$ is the distance from $r$ to $x$ and that $P(x)$ is a shortest path. ■

## 3.5 Algorithm for Longest Paths

To complement Dijkstra's algorithm for finding the short path, in this section we give an algorithm for finding the *longest* path between two vertices in a directed graph.

It is not immediately clear why we might want to do this, so first in Subsection 3.5.1 we give a motivational problem: scheduling work on a complicated project. The algorithm we present will only work on *acyclic* directed graphs, so in Subsection 3.5.2 we define these, explain why this isn't a restriction for our intended application, and give the first step of the algorithm: "topologically sorting" the vertices of an acyclic directed graph. Finally, in Subsection 3.5.3 we explain the actual algorithm.

### 3.5.1

The main application of the longest path algorithm is in scheduling. Suppose we have a large project -- say, building a house -- that is composed of many smaller projects: digging the foundation, building the walls, connecting to gas, electricity, and water, building the roof, doing the interiors, landscaping, etc.

Some of these activities will require others to be done before them (you can't put the roof on before you've built the walls; you don't want to do the landscaping before you've dug your water lines), while others could be done at the same time (finishing the interiors and doing the landscaping). Each sub-job has an expected amount of time required to finish it; you'd like to know before

hand how long the whole task will take, and when the various sub-jobs should be done so you can arrange the contractors.

From a series of jobs like this, we will construct a weighted, directed, acyclic graph. The edges will be the sub-jobs. The weights of each edge will be the expected length of time that job has. The structure of the graph will encode the dependencies of the subjobs on each other -- an edge $e$ will flow into an edge $f$ if the job $f$ immediately depends about the job $e$.

We will work out the construction of this graph in one example. It is not always trivial to construct the directed graph from the table of jobs and dependencies. It is not clear what the vertices should be, and sometimes dummy edges and vertices need to be encoded. You do not need to worry about constructing these graphs in general, though if you're curious it can be interesting to think about. Any exam question about this topic would supply you with the directed graph.Example === Consider the following table, listing tasks $$A-H$$, the expected time of completion for each task, and the required tasks before a given task can be started.

**Table 3.5.1**

Task Time PrerequisitesA 6 B 7 C 4 AD 3 AE 4 B,DF 10 CG 3 CH 10 E,G

Here is the corresponding graph encoding this information:

Construction of the graph ==== We outline how the graph above was constructed. We make one vertex for the start, one vertex for the finish, and then another vertex for each set of dependencies, that is, the entries in the third column. Then we draw an edge for each letter, beginning at the vertex corresponding to its set of prerequisites (or the start, if it has none), and ending at the vertex that contains it as a prerequisite (or the end, if no tasks require it as a prerequisite). Note that this method works only if any two sets of prerequisites either have nontrivial intersection or are identical. The tricky cases you don't have to worry about are when this isn't true. Longest Paths ---- With that detour out of the way, we see why finding the longest path in a directed acyclic graph is useful: in case the edges are tasks and the weights are expected times, the length of the longest path is the minimal time the whole project would be able to be completed. Moreover, it is useful to actually know what the longest paths are -- to achieve this minimal time, each task in the longest path must be completed in the expected amount of time, and the next task in the path must be started immediately when the first one finishes. For this reason, the longest paths are known as ∗critical paths∗.

### 3.5.2

### 3.5.3

## 3.6 The Traveling Salesperson Problem

In this section we discuss the Travelling Salesperson problem. In Subsection 3.6.1 we introduce the problem and give some explanation of why it is very hard in general. Rather than try to solve it exactly, we will resort to providing upper and lower bounds for the solution. In Subsection 3.6.2 we discuss various methods of constructing upper bounds. In Subsection 3.6.3 we give a method of constructing lower bounds.

### 3.6.1 Introduction to the Traveling Salesperson Problem

Let us first describe the Traveling Salesperson Problem, or TSP for short, in informal language, and then translate it into a question about graph theory.

Imagine you work for a company, travelling from city to city, trying to sell some product in each (for instance, encyclopedias). You are assigned a list of cities you need to visit, and you need to start from your home, travel from city to city visiting them all, and finally return to your home. Of course, travelling from city to city is expensive (either in terms of money, travel time, or something else), and to turn a profit your company wants you to organize the order you visit each of cities so that the total cost is as cheap as possible. This minimiziation problem is the TSP.

Translated into graph theory, the TSP can be succinctly stated as follows: given a weighted graph **G**, find the cheapest Hamiltonian path. That is, the cheapest closed walk on **G** that visits every vertex exactly once.

First, note that it is enough to consider the complete graph $K_n$. If we are given some other weighted graph **G**, we can add all the edges not in **G** but make their weights *much* larger than any of the weights inside **G**.

Another important point is that the problem of determining whether a given graph **G** has a Hamiltonian cycle is a special case of the traveling salesman problem. To see this, suppose we're given a graph **G**, and we want to determine whether it is Hamiltonian. We create a weighted $K_n$, with vertices the vertices of **G** by giving the edge $v - w$ a very small weight $\epsilon$ if $v$ and $w$ *are* adjacent in **G**, and a very large weight $M$ if $v$ and $w$ $*$are not$*$ adjacent in **G**. Then, any Hamiltonian path in **G** would have cost $n\epsilon$, where as any path that uses an edge not in **G** costs more than $M$. So, if we make $M > n\epsilon$, the TSP for our weighted $K_n$ will have a solution with cost less than $M$ if and only if **G** had a Hamiltonian cycle.

Since determining whether a graph **G** is Hamiltonian is difficult (NP complete), the TSP will also be. As such, we will not discuss any algorithms for actually solving TSP. Instead, we will discuss methods for giving upper and lower bounds for the TSP.

### 3.6.2 Finding upper bounds to the TSP

Getting *good* upper bounds to the TSP will turns out to be difficult. However, finding not so good upper bounds will turn out to be quite easy.

For instance, any solution to the TSP will be a Hamiltonian cycle, and in particular if **G** contains $n$ vertices, the TSP solution will contain $n$ edges. Let $M$ be the weight of the most expensive edge in **G**.

$$w(C) = \sum_{i=1}^{n} w(e_i) \leq \sum_{i=1}^{M} = nM$$

Since the TSP asks for the cheapeast Hamiltonian cycle, taking *any* Hamiltonian cycle and calculating its cost will be an upper bound for the TSP. Just choosing a random Hamiltonian cycle will in general be very expensive and silly -- for instance, going from Sheffield to London to Rotherham to Edinburgh to Chesterfield to Glasgow to Nottingham to Brighton is clearly not optimal.

A greedy algorithm will give a heuristically better result: we call it the *nearst neighbor algorithm*. At each step, simply go to the nearest city you have not already visited. This will give good results at the beginning, but since we do not do any planning ahead, it will in general give bad results, as the following example illustrates:

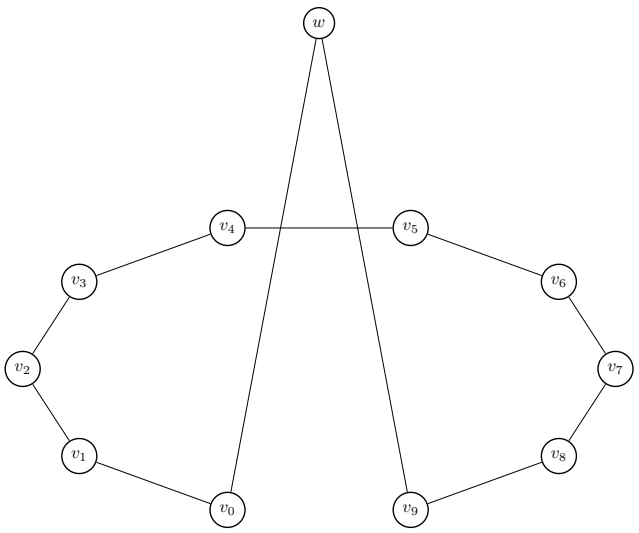



**Figure 3.6.1** The graph **G** where nearest neighbour struggles

Consider running the Nearest Neighbor algorithm starting at $v_0$. At the first step, we have a choice -- we could go to $v_1$ or to $v_9$. Suppose we go to $v_1$. After that, our choice is forced -- $v_1 - v_2 - v_3 - v_4 - v_5 - v_6 - v_7 - v_8 - v_9$ costs one at each step. Now, we still have to visit $T$ before returning to $V_0$, which will cost us 10 to detour through. We clearly should have planned ahead and visited $$T$$ in between vertices $v_4$ and $v_5$ at a cost of 4.

Clearly the nearest neighbour algorithm is not in general very good, and better algorithms are possible. We present it first to give a quick but reasonable way to get a solution to TSP that isn't *completely* horrible, and second to illustrate that greedy algorithms in general will not be efficient. We briefly mention two other ways to get lower bounds.

Another, slightly better, greedy algorithm might be called *nearest insertion*. It inductively builds bigger and bigger closed loops one vertex at time. When there is a closed loop with $k$ vertices $v_1 - v_2 - v_3 - \cdots - v_k - v_1$ and we want to add vertex $w$ to the loop, we look at each of the adjacent legs $v_i - v_{i+1}$, and determine how much it would raise the cost to insert the next vertex $w$ in between those two cities (changing the path to $v_1 - w - v_{i+1}$), being sure to also check for inserting it between $v_k$ and $v_1$. This does much better at our example above, but can run into other problems, and involves a little more bookkeeping and arithmetic, so I won't ask you to implement it on the exam.

Another method inolves a qualitive change of view. The greedy algorithms we describe so far are only heuristics to getting a decent path. There is no guarantee that they produce an output that is in any way close to the optimal path, and indeed examples can be engineered to make them extremely bad. It would be nice to have an upper bound that was guaranteed to not be *too* far off the optimal solution. The Christofides algorithm does just that, by producing a Hamiltonian cycle that is guaranteed to have weight at most 3/2 of the weight of the optimal solution. Very briefly, it does this by starting with a minimal weight spanning tree, makes a subgraph by adding edges to the tree until every vertex has even degree, taking an Eulerian circuit of that, and then removing edges to get a Hamiltonian cycle.

For nearly fifty years, Christofides algorithm was the best known guaranteed upper bound on the Travelling Salesperson problem, but in the summer 2020 of Nathan Klein, Anna Karlin and Shayan Oveis Gharan managed to modify the algorithm to give a very slight improvement, producing a cycle guaranteed to be within $3/2 - \varepsilon$ for some $\varepsilon > 10^{-36}$. See this Quanta article for a popular

account of their work.

### 3.6.3 A lower point for TSP

To get a lower bound for TSP we have to be a little more intelligent. Suppose we had a solution $C$ to the TSP for $\Gamma$, and that we deleted one vertex $v$ from $C$. Deleting a vertex from a cycle gives us a path $P$, and in particular a tree. Furthermore, $P$ visits every vertex in $\Gamma$ except for $v$, and so it is a spanning tree of $\Gamma \setminus v$.

We can use Kruskal's algorithm (or another) to find a minimal spanning tree $T$ of $\Gamma \setminus v$, and we have that $w(P) \geq w(T)$. The cycle $C$ contains just two more edges, from $v$ to two other vertices, say $a$ and $b$. We can obtain lower bounds on the weights of the edges $v - a$ and $v - b$ by taking the weights of the lowest two edges out of $v$, maybe $e_1$ and $e_2$. We have

$$w(C) = w(P) + w(a - v) + w(b - v) \geq w(T) + w(e_1) + w(e_2)$$

giving us a lower bound on solutions to the TSP.

## 3.7 Exercises

**1.** For the graph in Figure 3.7.1, use Kruskal's algorithm ("avoid cycles") to find a minimum weight spanning tree. Your answer should include a complete list of the edges, indicating which edges you take for your tree and which (if any) you reject in the course of running the algorithm.

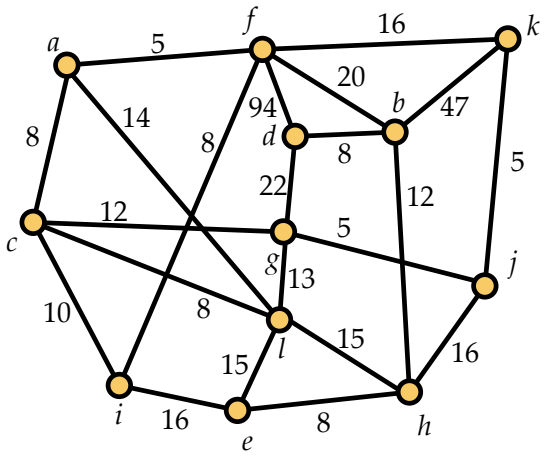



**Figure 3.7.1** Find a minimum weight spanning tree

**2.** For the graph in Figure 3.7.1, use Prim's algorithm ("build tree") to find a minimum weight spanning tree. Your answer should list the edges selected by the algorithm in the order they were selected.

**3.** For the graph in Figure 3.7.2, use Kruskal's algorithm ("avoid cycles") to find a minimum weight spanning tree. Your answer should include a complete list of the edges, indicating which edges you take for your tree and which (if any) you reject in the course of running the algorithm.

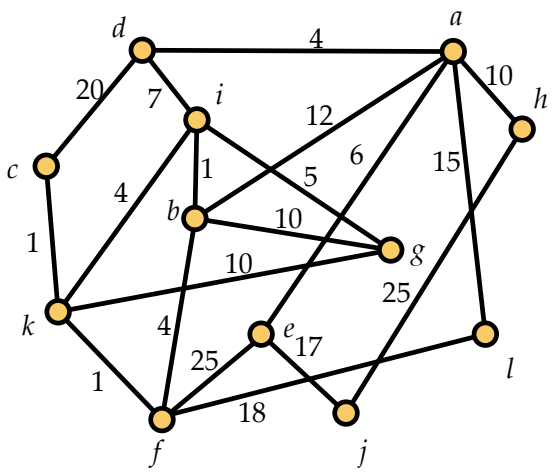



**Figure 3.7.2** Find a minimum weight spanning tree

**4.** For the graph in Figure 3.7.2, use Prim's algorithm ("build tree") to find a minimum weight spanning tree. Your answer should list the edges selected by the algorithm in the order they were selected.

**5.** For the graph in Figure 3.7.3, use Kruskal's algorithm ("avoid cycles") to find a minimum weight spanning tree. Your answer should include a complete list of the edges, indicating which edges you take for your tree and which (if any) you reject in the course of running the algorithm.

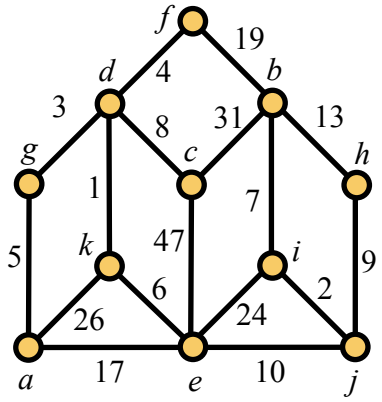



**Figure 3.7.3** Find a minimum weight spanning tree

**6.** For the graph in Figure 3.7.3, use Prim's algorithm ("build tree") to find a minimum weight spanning tree. Your answer should list the edges selected by the algorithm in the order they were selected.

**7.** A new local bank is being created and will establish a headquarters $h$, two branches $b_1$ and $b_2$, and four ATMs $a_1$, $a_2$, $a_3$, and $a_4$. They need to build a computer network such that the headquarters, branches, and ATMs can all intercommunicate. Furthermore, they will need to be networked with the Federal Reserve Bank of Atlanta, $f$. The costs of the feasible network connections (in units of \$10,000) are listed below:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $hf$ | 80 | $hb_1$ | 10 | $hb_2$ | 20 | $b_1b_2$ | 8 |
| $fb_1$ | 12 | $fa_1$ | 20 | $b_1a_1$ | 3 | $a_1a_2$ | 13 |
| $ha_2$ | 6 | $b_2a_2$ | 9 | $b_2a_3$ | 40 | $a_1a_4$ | 3 |
| $a_3a_4$ | 6 | | | | | | |

The bank wishes to minimize the cost of building its network (which must allow for connection, possibly routed through other nodes, from each node to each other node), however due to the need for high-speed communication, they ***must*** pay to build the connection from $h$ to $f$ as well as the connection from $b_2$ to $a_3$. Give a list of the connections the bank should establish in order to minimize their total cost, subject to this constraint. Be sure to explain how you selected the connections and how you know the total cost is minimized.

**8.** A disconnected weighted graph obviously has no spanning trees. However, it is possible to find a spanning forest of minimum weight in such a graph. Explain how to modify both Kruskal's algorithm and Prim's algorithm to do this.

**9.** Prove Proposition 3.2.3.

**10.** In the paper where Kruskal's algorithm first appeared, he considered the algorithm a route to a nicer proof that in a connected weighted graph with no two edges having the same weight, there is a *unique* minimum weight spanning tree. Prove this fact using Kruskal's algorithm.

**11.** Use Dijkstra's algorithm to find the distance from $a$ to each other vertex in the digraph shown in Figure 3.7.4 and a directed path of that length.

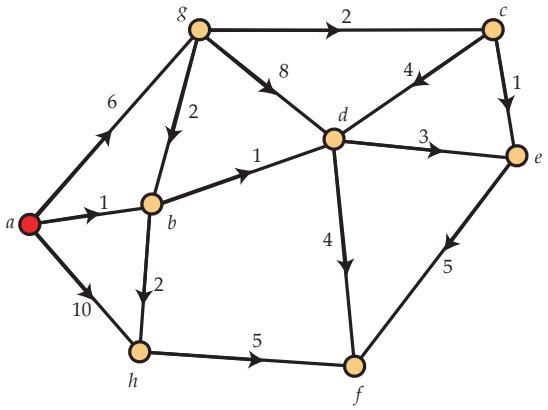



**Figure 3.7.4** A directed graph

**12.** Table 3.7.5 contains the length of the directed edge $(x, y)$ in the intersection of *row* $x$ and *column* $y$ in a digraph with vertex set $\{a, b, c, d, e, f\}$. For example, $w(b, d) = 21$. (On the other hand, $w(d, b) = 10$.) Use this data and Dijkstra's algorithm to find the distance from $a$ to each of the other vertices and a directed path of that length from $a$.

**Table 3.7.5 A digraph represented as a table of data**

| $w$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |
|---|---|---|---|---|---|---|
| $a$ | 0 | 12 | 8 | 43 | 79 | 35 |
| $b$ | 93 | 0 | 18 | 21 | 60 | 33 |
| $c$ | 17 | 3 | 0 | 37 | 50 | 30 |
| $d$ | 85 | 10 | 91 | 0 | 17 | 7 |
| $e$ | 28 | 47 | 39 | 14 | 0 | 108 |
| $f$ | 31 | 7 | 29 | 73 | 20 | 0 |

**13.** Use Dijkstra's algorithm to find the distance from $a$ to each other vertex in the digraph shown in Figure 3.7.6 and a directed path of that length.

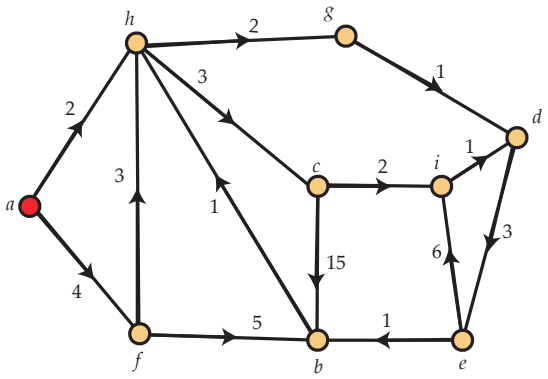



**Figure 3.7.6** A directed graph

**14.** Table 3.7.7 contains the length of the directed edge $(x, y)$ in the intersection of *row* $x$ and *column* $y$ in a digraph with vertex set $\{a, b, c, d, e, f\}$. For example, $w(b, d) = 47$. (On the other hand, $w(d, b) = 6$.) Use this data and Dijkstra's algorithm to find the distance from $a$ to each of the other vertices and a directed path of that length from $a$.

**Table 3.7.7 A digraph represented as a table of data**

| $w$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ |
|---|---|---|---|---|---|---|
| $a$ | 0 | 7 | 17 | 55 | 83 | 42 |
| $b$ | 14 | 0 | 13 | 47 | 27 | 17 |
| $c$ | 37 | 42 | 0 | 16 | 93 | 28 |
| $d$ | 10 | 6 | 8 | 0 | 4 | 32 |
| $e$ | 84 | 19 | 42 | 8 | 0 | 45 |
| $f$ | 36 | 3 | 76 | 5 | 17 | 0 |

**15.** Give an example of a digraph having an *undirected* path between each pair of vertices, but having a root vertex $r$ so that Dijkstra's algorithm cannot find a path of finite length from $r$ to some vertex $x$.

**16.** Notice that in our discussion of Dijkstra's algorithm, we required that the edge weights be nonnegative. If the edge weights are lengths and meant to model distance, this makes perfect sense. However, in some cases, it might be reasonable to allow negative edge weights. For example, suppose that a positive weight means there is a cost to travel along the directed edge while a negative edge weight means that you make money for traveling along the directed edge. In this case, a directed path with positive total weight results in paying out to travel it, while one with negative total weight results in a profit.

(a) Give an example to show that Dijkstra's algorithm does not always find the path of minimum total weight when negative edge weights are allowed.

(b) Bob and Xing are considering this situation, and Bob suggests that a little modification to the algorithm should solve the problem. He says that if there are negative weights, they just have to find the smallest (i.e., most negative weight) and add the absolute value of that weight to every directed edge. For example, if $w(x, y) \geq -10$ for every directed edge $(x, y)$, Bob is suggesting that they add 10 to every edge weight. Xing is skeptical, and for good reason. Give an example to show why Bob's modification won't work.

# Chapter 4

# Graphs on Surfaces

This chapter covers drawing graphs on surfaces. To motivate this topic, we will begin by thinking about videogames

We start with discussing whether or not graphs are planar, proving that $K_{3,3}$ and $K_5$ are not planar using a method we call the Planarity Algorithm for Hamiltonian graphs. We discuss the more general Kuratowski's theorem for proving any graph is planar or not. We introduce other surfaces, and how to draw graphs on them -- the sphere, Mobius band, and torus in particular. After a brief discussion of dual graphs, we prove Euler's theorem about planar graphs and explore several applications.

## 4.1 Introduction to Graphs on Surfaces

We begin our study of graphs on surfaces with an old chestnut of a problem, the solution of which we will develop into a more general algorithm.

### 4.1.1 The Utilities Problem

Suppose there are three houses, owned by Alice, Bob, and Carol, and they'd each like to be connected to one of three Utilities, say, gas, electricity, and water. There is no real difficulty in the real world, but if we add the restriction that we don't want any of the lines to cross over or under each other, the problem becomes quite interesting. A failed attempt at drawing a solution is shown here.

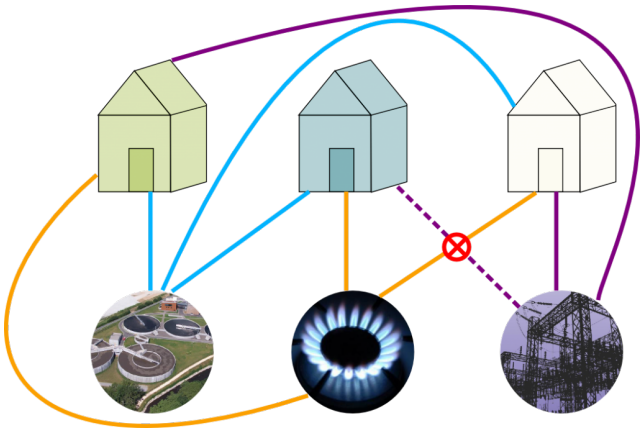


**Figure 4.1.1** An attempt at solving the three utilities problem

Although this attempt failed, it seems very difficult to rule out that some other attempt wouldn't succeed; trying to make a case by case argument seems quite difficult to organize, and it's not clear that there are even finitely many possibilities. We need a careful way to approach the problem, which we will do in a moment, but first we will use this problem as motivation to make a few definitions.

## 4.1.2

**Definition 4.1.2** A graph is *planar* if it can be drawn on a piece of paper so that no edges cross. ◊

That definition is a bit loose -- for instance, it's left implicit, we're drawing the edges as lines, with the endpoints being the two vertices it connects. But this will be strong enough for our purposes.

With this definition in hand, the Utilities Question is asking whether the graph $K_{3,3}$ is planar -- treat the three utilities as red vertices, say, and the three houses as the blue vertices. This doesn't really help us organize our proof, however. To do that, we will use the basic fact that any circle drawn on the plane has an inside and an outside.

This last fact sounds absolutely trivial, but first, it is not true on other surfaces, for instance, on the torus -- in our video game world, the top of the screen makes a circle, but a point just above this circle is really at the bottom of the video game world, and so the circle doesn't cut the torus into two pieces; I also illustrated this with the Mobius band: the central line down the middle doesn't separate it into two pieces. This fact is usually stated as follows:

**Theorem 4.1.3 Jordan Curve Theorem.** *Any simple closed plane curve has an interior and an exterior*

Though easy to state, and intuitively obvious, the Jordan Curve Theorem is surprisingly subtle and difficult to prove; we won't use any more topology than this.

Before seeing it in practice, let's discuss informally how the Jordan Curve Theorem can be used to help prove whether a graph $G$ is planar or not. Suppose that we have found a large cycle $C_k$ as a subgraph of $G$. Then, if we had a planar drawing of $G$, this cycle would have to appear as a circle. By the Jordan Curve Theorem, this circle would have an inside and an outside, and every vertex and edge not in our cycle $C_k$ would have to be either entirely within the circle, or entirely outside the circle. This gives us a way to organize the case by case argument.

The bigger a cycle we can find, the fewer other vertices and edges we need to consider, and so we have a much cleaner case by case argument. In the best cases, the graph is Hamiltonian and the cycle $C_k$ includes all the vertices of $G$, and we only have to do a case by case analysis for some the remaining edges.

Let's see how this general principle gets illustrated in practice

**Theorem 4.1.4** *$K_{3,3}$ isn't planar*

*Proof.* First let's name the vertices of $K_{3,3}$: let the vertices $a, b, c$ be the blue circle vertices, and $x, y, z$ be the red rectangle vertices. Then the path $a - x - b - y - c - z - a$ is a Hamiltonian cycle, and so if $K_{3,3}$ were planar it would be drawn as a circle in the plane, as shown below:

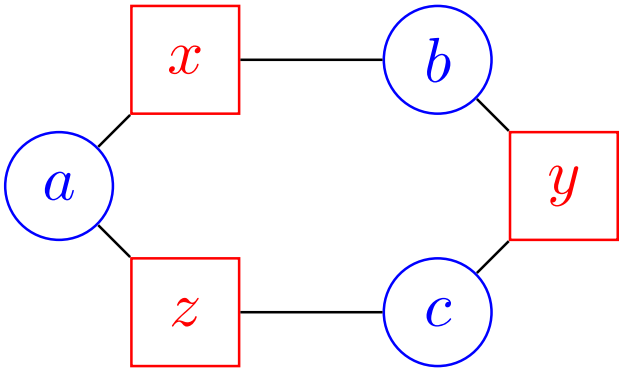



**Figure 4.1.5** The Hamiltonian cycle in $K_{3,3}$

This contains 6 of the 9 edges of $K_{3,3}$; we need to add the edges $a - y, b - z$ and $c - x$. The edge $a - y$ could be drawn inside the circle or outside, suppose we draw it inside, as shown below, with the added edge dashed.

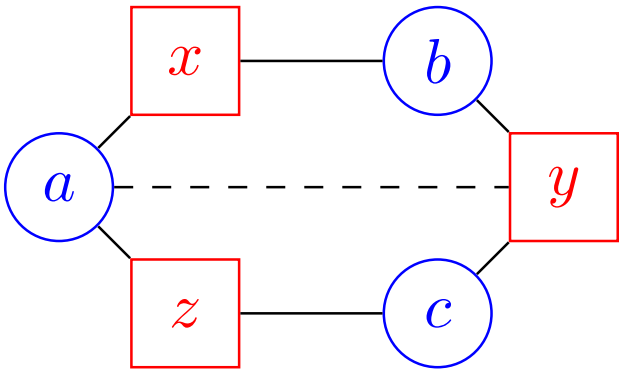



**Figure 4.1.6** Adding $a - y$ inside

Then on the inside of the circle, $x$ and $c$ are on different sides of the line $a - y$, and so the edge connecting them must go outside the circle. The added edge could go around the right of the circle, as shown below here:

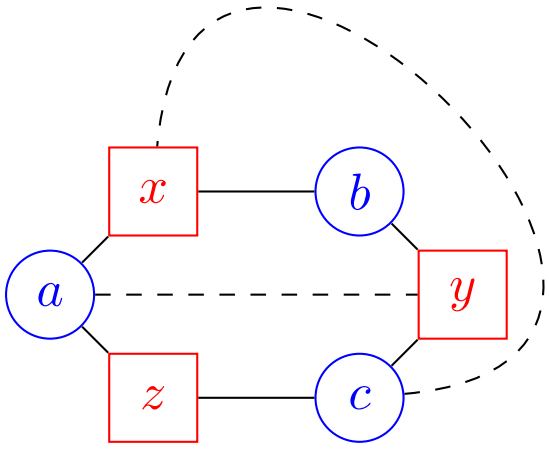



**Figure 4.1.7** Adding $a - y$ inside

or around the left, as shown here:

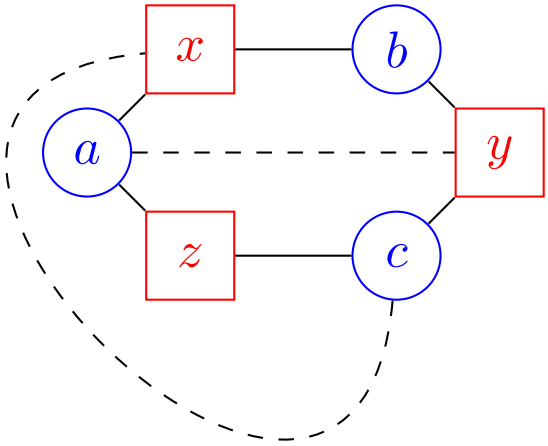



**Figure 4.1.8** Adding $a - y$ inside

But now $b$ and $z$ are different sides of $a - y$ inside the circle, and on different sides of $c - x$ outside the circle, and so cannot be connected without making two edges cross.

If we had began by drawing $a - y$ outside the circle, then we would have had to draw $c - x$ inside the circle, and had the same problem with being able to draw the last line; as shown here:

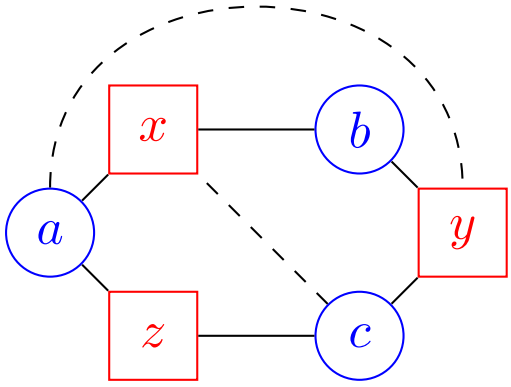



**Figure 4.1.9** Adding $a - y$ inside

■

## 4.2 The planarity algorithm for Hamiltonian graphs

In the previous chapter we showed that $K_{3,3}$ isn't planar; in this section we investigate how the ideas we used to solve the utilities problem for $K_{3,3}$-- namely, the Jordan Curve theorem and the fact that $K_{3,3}$ is Hamiltonian -- generalize to other graphs. In the end, this will culminate in "The Planarity Algorithm for Hamiltonian Graphs".

### 4.2.1 Stereographic Projection and Unnecessary cases

It will make our life easier if before we investigate other graphs we streamline our proof for $K_{3,3}$ slightly: there were a few times where we had to treat different cases that wound up behaving essentially the same, and we'd like to see that we didn't actually need to treat them as separate cases. In particular, we would like to show that the following three seemingly different ways to connect the first two vertices lead to the same analysis:

1. Connecting them inside the Hamiltonian cycle
2. Connecting them outside "to the left"
3. Connecting them outside "to the right"

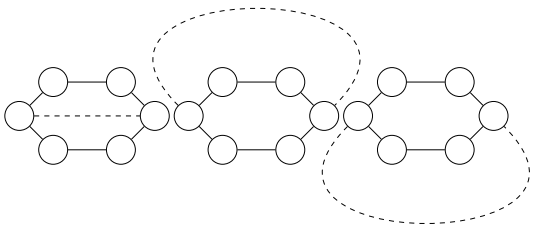


**Figure 4.2.1** The three cases

The solution will be to think about drawing the graphs on the sphere $S^2$ instead of the plane. First, let's see why this solves our problem. On the plane, the inside of a circle is different from the outside of a circle -- the inside is bounded, but the outside is unbounded. However, on the sphere, the two sides of a circle are equivalent -- you can deform any circle to be an equator, and then the northern hemisphere is equivalent to the southern hemisphere. This shows on there sphere, the inside and the outside aren't really different cases.

Furthermore, going around the outside to "the left" or "to the right" are equivalent on the sphere -- you can slowly make the path around the sphere bigger and bigger, and then slip it around the north or south pole, and back. Alternatively, we've already seen that the inside of the circle is equivalent to the outside of the circle on the sphere $S^2$, and on the inside of the circle it doesn't matter exactly how the two points are connected, and so it shouldn't matter on the outside, either.

So we've argued that if we're trying to draw a graph on the sphere, all three cases are the same, but it should still feel like a bait-and-switch: we weren't trying to draw graphs on there sphere, we were trying to draw graphs on the plane. The connection comes from the fact that the sphere can be viewed as a plane with one additional point.

**Proposition 4.2.2** *Let $p \in S^2$ be any point. Then $S^2 \setminus \{p\} \cong \mathbb{R}^2$.*

*Proof.* One way to visualize this is imagine the sphere as being made from very flexible clay. If we poke a small hole in the top of the sphere, we could stick our fingers in and make the hole larger, and gradually stretch and bend and reform for the sphere to be a flat disk, which could be stretched to be the whole plane, in the same way the tangent function maps the interval $(-\pi/2, pi/2)$ to the whole real line $\mathbb{R}$

Alternatively, one could use stereographic projection, as shown in Figure . Draw $S^2$ in $\mathbb{R}^3$ as the unit sphere at the origin, and let $N = (0, 0, 1)$ be the north pole of the sphere. Stereographic projection gives a bijection between $S^2 \setminus \{N\}$ (the sphere minus the north pole) to the plane, as follows: for any point $p \neq N$ the line through $p$ and $N$ must meet the $xy$-plane at one point. On the other hand, any line through $N$ and a point on the $xy$-plane must meet the sphere at one other point.

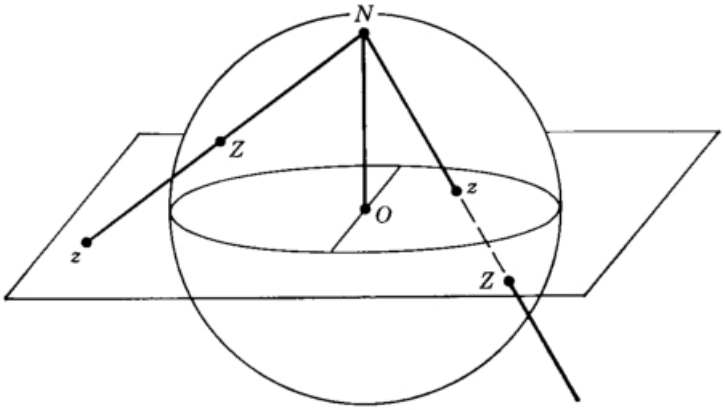



**Figure 4.2.3** Stereographic Projection

Accepting that $S^2$ is $\mathbb{R}^2$ minus one point, we see that we can draw a given graph $\mathbf{G}$ on $S^2$ if and only if we can draw $\mathbf{G}$ on $\mathbb{R}^2$: if we draw it on $\mathbb{R}^2$, we can view the $\mathbb{R}^2$ as a small patch of $S^2$. And if we have a drawing on $S^2$, there must be at least one point on $S^2$ that isn't in the drawing of $G$, and doing

stereographic projection from that point gives a drawing of $G$ on the plane $\mathbb{R}^2$. ■

### 4.2.2 The planarity algorithm

The preceeding discussion may have felt heavy going, but the upshot is that the cases that seemed "the same" in our analysis of $K_{3,3}$ actually are the same, and similar cases will be the same for any graph. This will make it much easier to extend our reasoning to more complicated graphs.

Suppose that $G$ is Hamiltonian, and choose a Hamiltonian cycle; if $G$ were planar than this cycle must be drawn as a circle, and every other edge must either lie entirely inside or entirely outside the graph. Now consider two edges $e = ab$ and $f = xy$ that are not part of the cycle. Depending on the order that $a, b, x$ and $y$ appear as you go around the Hamiltonian cycle, one of two things will happen:

1. If the vertices of $e$ and $f$ do not interlace (e.g. $abxy, yxab, xbay, \cdots$), or if they share a vertex (e.g., $a = x$), then $e$ may be drawn both inside or both outside the circle without crossing

2. If the vertices of $e$ and $f$ do interlace (e.g. $axby, xayb, yaxb, \ldots...$) then if $e$ and $f$ are drawn both inside or both outside the circle, they must cross

This motivates the following definition

**Definition 4.2.4 Cross($G, C$).** Let $G$ be a Hamiltonian graph, and $C$ a Hamiltonian cycle in $G$. The *crossing graph* of $G$ and $C$, denoted Cross$(G, C)$ has as vertices the edges of $G$ that aren't in the cycle, and an edge between vertices $p$ and $q$ if the vertices of the corresponding edges interleave -- that is, $p$ and $q$ are adjacent if they cannot be drawn on the same side of the cycle $C$ without crossing. ◊

**Algorithm 4.2.5 The planarity algorithm for complete graphs.** *Suppose that $G$ is Hamiltonian, and $C$ is a Hamiltonian cycle. Then $G$ is planar if and only if Cross$(G, C)$ is bipartite.*

The idea is that if $G$ is planar, the vertices of Cross$(G, C)$ are naturally bicolored, with the red vertices, say, corresponding to the edges that are drawn inside the cycle $C$, and the blue edge corresponding to the edges that are drawn outside the cycle $C$. The definition of the edges of Cross$(G, C)$ guarantees there are no edges between vertices of the same color.

Similarly, if we can find a colouring of the vertices of Cross$(G, C)$ so that adjacent vertices have different colours, then we can draw all the edges of $G$ corresponding to red vertices of Cross$(G, C)$ inside (or outside) $C$ without having any of them cross.

**Example 4.2.6 The complete graph $K_5$ isn't planar.** Let's apply the planarity algorithm to the complete graph $K_5$. Let's label the vertices 1-5, and take our Hamiltonian cycle $C$ to be 123451, which we've drawn as the outside pentagon in the following figure:

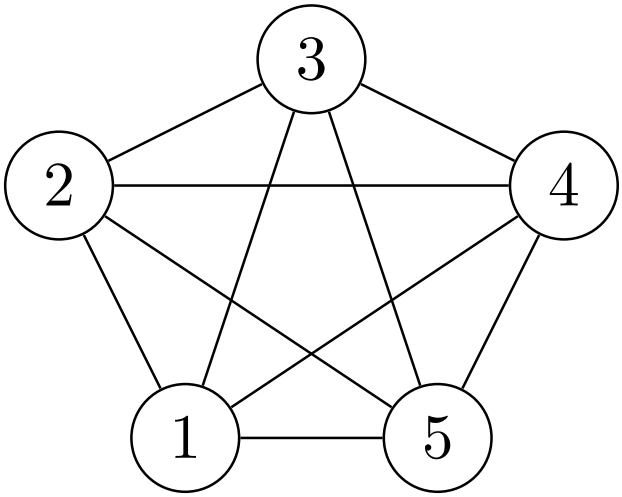



**Figure 4.2.7** The graph ($K_5$)

Since $K_5$ has $\binom{5}{2} = 10$ vertices, there are 5 edges that aren't used in $C$, namely 13,14, 24,25, and 35. So Cross($K_5, 123451$) will consist of these five vertices. We see that 13 will be adjacent to 24 and 25, since these edges would cross if drawn inside, but 13 is not adjacent to 14 or 35, since these edges would cross 13 if drawn on the same side of the circle, as illustrated in the next figure

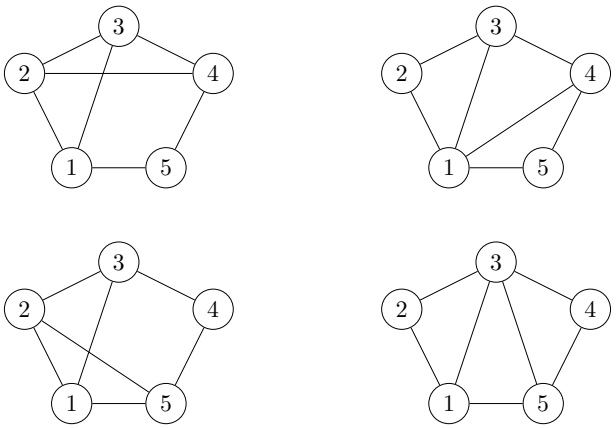



**Figure 4.2.8** The edges 13 does and does not cross

Similar consideration with the other edges show that Cross($K_5, 123451$) is the following graph, which is isomorphic to a five cycle:

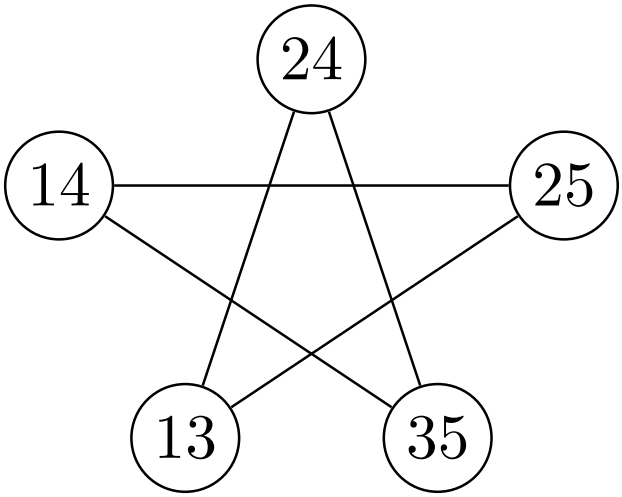



**Figure 4.2.9** The graph Cross($K_5$, 123451)

In particular, Cross($K_5$, 123451) is not bipartite. Hence, by the planarity algorithm for Hamiltonian graphs, we see that $K_5$ is not planar. □

**Example 4.2.10 A planar graph.** Let's use the planarity algorithm for Hamiltonian graphs to find a planar drawing of the graph shown in the next figure.

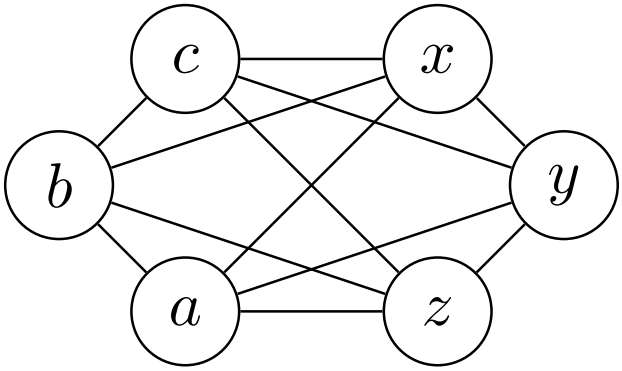



**Figure 4.2.11** A graph $H$

We see that $H$ is Hamiltonian and take as our Hamiltonian cycle the path around the outside, namely $abcxyza$. There are then six edges not contained in the Hamiltonian cycle, and we find that Cross($H, abcxyz$) is as follows:

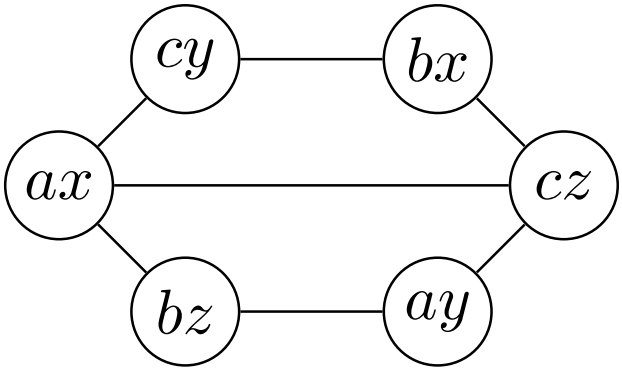



**Figure 4.2.12** The graph Cross($H, abcxyza$)

For instance, in $H$ the edge $ax$ crosses the three edges $cy, cz$ and $bz$, and so in Cross$(H, abcxyza)$, the vertex $ax$ is adjacent to these vertices.

The graph Cross$(H, abcxyza)$ has no odd cycles and hence is bipartite -- for instance, we may color $ax, bx$ and $ay$ red, and the other three vertices blue. Then, to draw $H$ in the plane without edges crossing, we draw the red edges inside the cycle, and the blue edges outside the cycle:

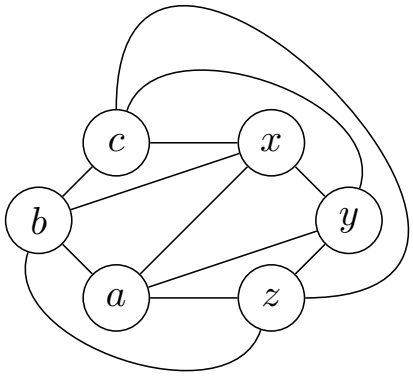



**Figure 4.2.13** The graph $H$ drawn without edges crossing

□

## 4.3 Kuratowski's Theorem

The planrity Algorithm for Hamiltonian graphs gives a very convenient and systematic way to determine whether a Hamiltonian graph is planar or not, and we saw that with some work it can be hacked to work for graphs that are "almost" Hamiltonian -- that have a cycle that go through all but one or two vertices, say.

Stretching these ideas further, the general logic of considering cycles and applying the Jordan Curve theorem to them would provide a way to prove whether an abritrary graph is planar or not, but as we have more or more vertices that aren't on our cycle to consider the arguments would get more and more complicated, and it's clear that a better method is desirable. In this section we will present, (but not completely prove) Kuratowski's theorem, which gives a method to determine whether or not an arbitrary graph is planar.

### 4.3.1 Planarity, subgraphs, and subdivisions

The idea behind Kuratowski's theorem rests on two small observations, which we illustrate in a simple example before discussing more formally.

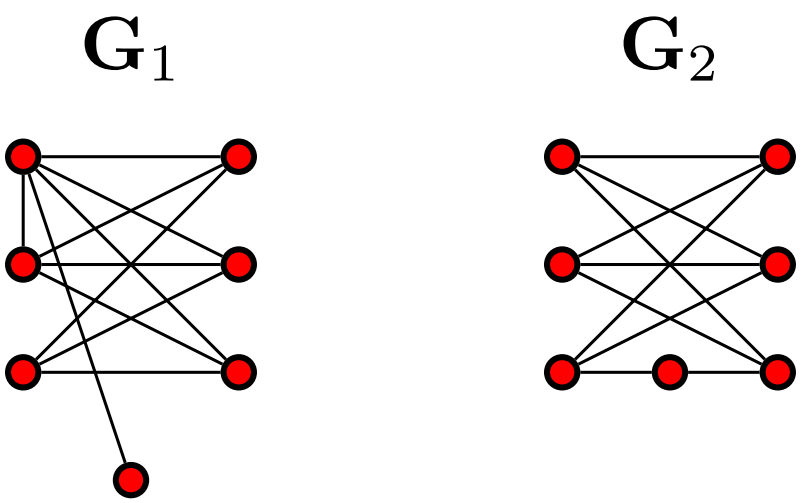



**Figure 4.3.1** Two nonplanar graphs

The graphs $\mathbf{G}_1$ and $\mathbf{G}_2$ in Figure 4.3.1 both *look* a lot like $K_{3,3}$, and since $K_{3,3}$ is nonplanar, we might expect them to be nonplanar as well, but we need to be careful and precise in checking this. We work this out in the next example.

**Example 4.3.2 Two nonplanar graphs.** The graph $\mathbf{G}_1$ in Figure 4.3.1 is $K_{3,3}$ with a vertex and two edges added to it; put another way, $K_{3,3}$ is a subgraph of $\mathbf{G}_1$. If we could draw $\mathbf{G}_1$ in the plane, we could just ignore this extra vertex and these two edges, and we'd have a drawing of $K_{3,3}$ in the plane, but we know $K_{3,3}$ isn't planar. So to avoid contradictions we are forced to conclude that $\mathbf{G}_1$ isn't planar.

The graph $\mathbf{G}_2$ looks just like $K_{3,3}$, but we have added an extra vertex of degree two in the middle of one of the edges. Note that $K_{3,3}$ is *not* a subgraph of $\mathbf{G}_2$, and so we need to use slightly different reasoning than we did for showing $\mathbf{G}$ isn't planar. But drawing $\mathbf{G}_2$ is just like drawing $K_{3,3}$ and then adding an extra dot for the new vertex of degree two. If we could draw $\mathbf{G}_2$ in the plane, we could just skip adding this extra dot, and we'd have a drawing of $K_{3,3}$ in the plane. Again, since we know $K_{3,3}$ isn't planar, we see that $\mathbf{G}_2$ isn't planar, either. □

We will now generalize the methods we used to show $\mathbf{G}$ and $\mathbf{H}$ are nonplanar and summarize them as lemmas. The reasoning we used to prove that $\mathbf{G}$ was nonplanar doesn't need to be changed at all to prove:

**Lemma 4.3.3** *If* $\mathbf{H}$ *is nonplanar, and* $\mathbf{H}$ *is a subgraph of* $\mathbf{G}$*, then* $\mathbf{G}$ *isn't planar.*

*Proof.* We've essentially already proved it, but we'll restate the reasoning in a different way for completeness.

When we draw a graph $\mathbf{G}$ in the plane, we also draw all the subgraphs of $\mathbf{G}$ in the plane. Thus, if $\mathbf{G}$ is planar, then all of its subgraphs are planar. Our lemma is the contrapositive of this statement. ■

Lemma 4.3.3 is logically equivalent to the discussion above, but it's worth restating the logic in this direction as well: if we can't draw $\mathbf{G}$ in the plane, then we certainly can't draw $\mathbf{H}$ in the plane without edges crossing, as if we could then we'd have a drawing of $\mathbf{H}$ as part of our big drawing.

**Example 4.3.4 Complete graphs.** If $n > 5$, then $K_n$ is not planar by Lemma 4.3.3, because $K_5$ is a subgraph of $K_n$, and we know that $K_5$ isn't planar.

We could also have used the fact that $K_{3,3}$ is a subgraph of $K_n$, and $K_{3,3}$ is also nonplanar. □

To generalize the method we used to prove $\mathbf{G}_2$ is non-planar, we first make a form definition that encapsulates the idea of "adding dots" to the middle of edges:

**Definition 4.3.5 Subdivision.** We say that **G** is a *subdivision* of **H** if **G** is obtained from **H** by adding some vertices of degree two in the middle of some of the edges of **H**. ◊

**Example 4.3.6 Cycles and Paths.** Any path graph $P_n, n \geq 2$ is a subdivision of the graph $P_2$ consisting of two vertices with an edge between them. Any cycle graph $C_m, m \geq 3$ is a subdivision of the triangle $C_3$ □

**Lemma 4.3.7** *If* **G** *is a subdivision of a nonplanar graph* **H**, *then* **G** *is nonplanar.*

*Proof.* Suppose that **G** was planar, and draw it in the graph. Then erase the, vertices of degree we added when we subdivided **H**, merging the edges on either side to one. We obtain a planar drawing of **H**, a contradiction, and so **G** must not have been planar. ■

### 4.3.2 Kuratowski's Theorem

The definitions and lemma of the previous section essentially prove the "easy" direction of the following theorem, which will be our main tool for proving graphs aren't planar.

**Theorem 4.3.8 Kuratowski's Theorem.** *A graph* **G** *is nonplanar if and only if* **G** *has a subgraph that's a subdivision of* $K_{3,3}$ *or* $K_5$.

*Proof.* We will only prove one direction: that if **G** has such a subgraph, then **G** is nonplanar; the other direction is more difficult.

Suppose that **H** is a subgraph of **G** that is subdivision of $K_{3,3}$ or $K_5$. Since we've proven $K_{3,3}$ and $K_5$ are nonplanar, we know **H** is nonplanar by Lemma 4.3.7. Now since we **H** is a subgraph of **G** and we know **H** is nonplanar, we know **G** is nonplanar by Lemma 4.3.3. ■

Although we've only proven one direction of Kuratowski's theorem, it's the important direction -- the one that lets us prove graphs are nonplanar. The other direction would tell us information about subgraphs of a graph that we already knew was nonplanar for some other reason. Or, taking the contrapositive, it would let us prove a graph *was* planar by looking at all subgraphs of it and showing none of them looked like $K_5$ or $K_{3,3}$. But this would be a lot of work and there's a much easier way to show a graph is planar: draw it in the plane! If you're asked to prove a graph is planar, you will almost always also be asked to draw it in the plane.

However, we will *implicitly* use the hard direction in the following way: if a graph **G** is nonplanar, you can *always* use Kuratowski's theorem to prove that it's nonplanar. This is reassuring because it means our tool will always work to prove it's nonplanar, and that you aren't wasting your time looking for subgraphs that don't exist.

### 4.3.3 Applying Kuratowski's Theorem

The tricky part of using Kuratowski's theorem is actually *finding* the desired subgraph. We won't really discuss algorithm aspects of this; for any graph you are asked to prove non-planar, it will be possible to do so by educated trial an error. A few rules of thumb may be helpful, however. First, note

that subdivision cannot increase the degree of any vertex. So, for **G** to have a subgraph that's a subdivision of $K_5$, **G** has to have at least 5 vertice with degree at least 4; if it doesn't, but we still suspect **G** to be nonplanar, we know instead that we should be looking for a subdivision of $K_{3,3}$.

Conceptually, it can be useful to think that some vertices of **G** are going to be vertices of your $K_5$ or $K_{3,3}$, and we're going to need to connect those. We can use the remaining vertices of **G** as parts of subdivided edges between these, but these extra vertices can only be used in at most one such connection. Thus, these extra vertices are a limited resource we have, and a useful heuristic in looking for subgraph is to take a "greedy" approach, where we choose our vertices to require as few subdivisions as possible to make connections. We illustrate this idea in the next example.

**Example 4.3.9 The Petersen graph isn't planar.**

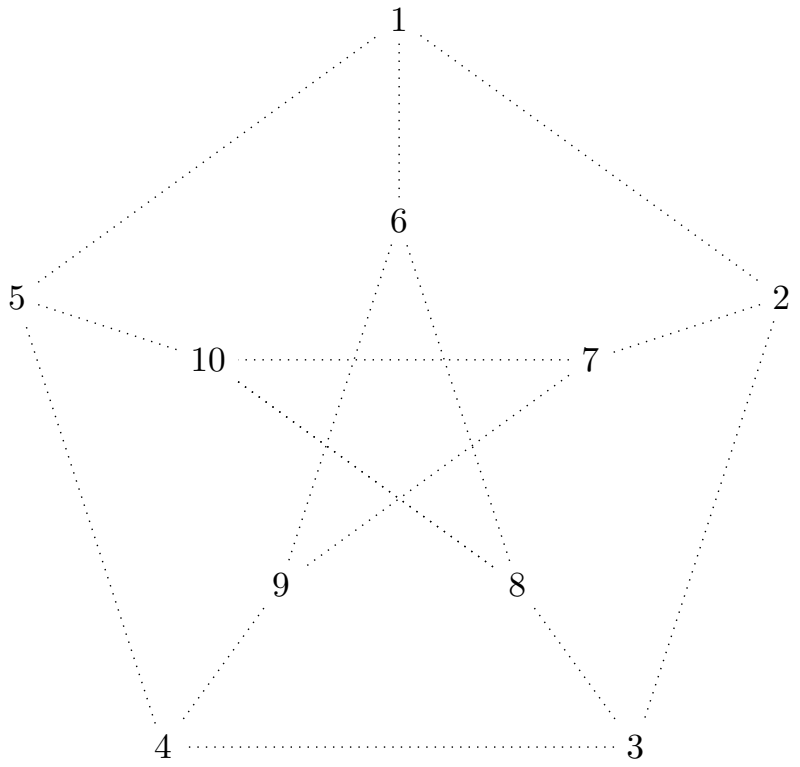



**Figure 4.3.10** The Petersen graph, labeled

Let us use Kuratowski's Theorem to prove that the Petersen graph isn't planar; Figure 4.3.10 has a drawing of the Petersen graph with the vertices labeled for referece. Since the Petersen graph is regular of degree three, we know that it can't have a subgrpah that's a subdivision of $K_5$, as it would need to have some vertices of degree 4 or larger.

It makes sense to attempt a greedy algorithm -- in the standard drawing of the Petersen grpah, pick the very top vertex 1 to be "red" and the three vertices adjacent to it to, 2, 5, and 6, to be "blue". We need two more red vertices. All vertices left are adjacent to exactly one blue vertex, so from a greedy point of view there is no preference for which vertex we pick to be the next blue vertex. Let us pick 9 to be another red vertex. Then it is connected directly to blue vertex 6, but we must find paths from 9 to 2 and 5. We could, for instance, take the path 9-4-3-2 to connect to 2, but that would use two vertices up while

the path 9-7-2 only uses one extra vertex, and so seems better. Then we can connected 9 to 5 through vertex 4, and vertex 9 has been connected to all the blue vertices.

Now, we need to choose one more vertex to be a red vertex, and the vertices we haven't used are 3, 10, and 8. If we tried to make 3 the last red vertex we run into a problem: we need to connect vertex 3 to 3 other vertices, but one of the edges goes to vertex 4 which was one of the subdivided vertices that we can't visit again. Hence, we only have two possible paths out of 3, and will ever only be able to connect it to two blue vertices. A similar problem occurs if we try to make 10 the last red vertex -- it's adjacent to the vertex 7 used as a subdivided vertex. The remaining choice is vertex 8, which works, as shown in the following diagram.

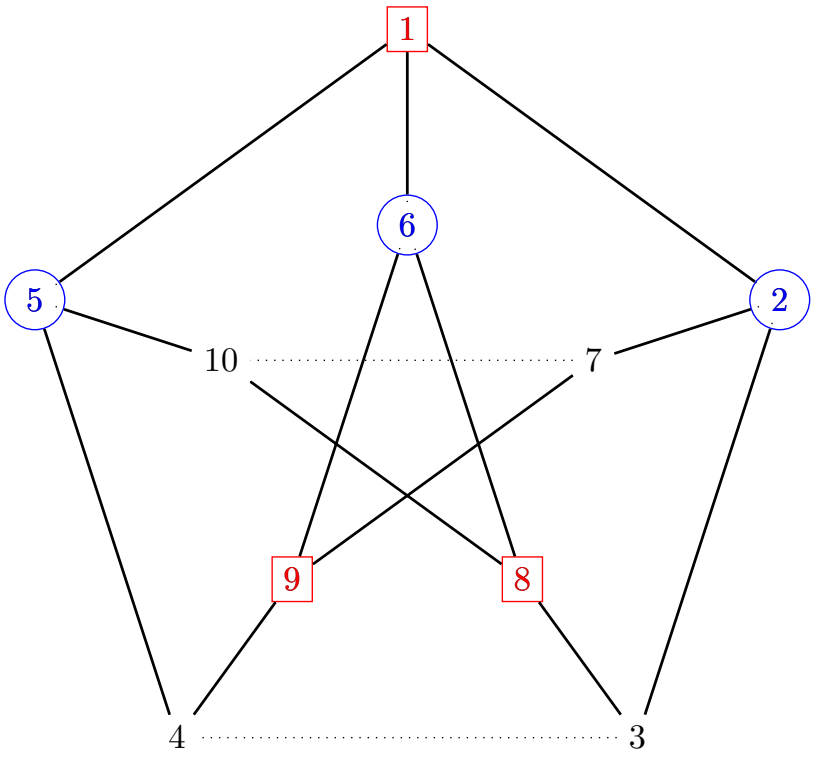



**Figure 4.3.11** A subdivided $K_{3,3}$ in the Petersen graph

The red and blue vertices of the subdivided $K_{3,3}$ are shown as squares/ circles, and the edges of the subdivided $K_{3,3}$ are colored thick -- only the dotted edges 7-10 and 3-4 of the Petersen graph are not used in the subgraph. □

## 4.4 Drawing Graphs on Other surfaces

We saw, using stereographic projection, that being able to draw a graph on the sphere is the same as being able to draw the graph on the plane. In this section we will discuss drawing graphs on other surfaces -- the torus and the Möbius strip we will discuss in detail, though similar ideas work for any surface. We need a way to represent such graphs on a piece of paper, for use in a book (or on the exam, say). Much of the material from the rest of this chapter (Kuratowski's theorem, Euler's theorem) have analogues for other surfaces, but are beyond the scope of this module.

### 4.4.1 Motivation and culture: Manifolds and Surfaces

In this short subsection we are going to be slightly informal. The goal is simply to motivate this section about drawing graphs on surfaces other than the sphere, and to give a motivating problem that we will solve the next section on Euler's Theorem.

People used to think that the earth was flat, because if you can't see the whole thing, but can just look at just one little patch of it around you, it looks like a piece of $\mathbb{R}^2$. Formally, a mathematicians say "locally homeomorphic to $\mathbb{R}^2$" to mean that you can't tell it apart from $\mathbb{R}^2$ by just looking at a small piece of it.

To compliment the familiar earth idea of the earth being round, we give a few more shocking thought-experiment examples. The first is: how do we know the earth is round? I, personally haven't been to space, haven't been all over the world. Maybe there's a giant tunnel running from the south pole to the north pole, and the earth is really a *torus* (the surface of a donut or a bagel).

The idea of the earth being the surface of the torus probably seems absolutely ridiculous, but it happens "by accident" in videogames. In old games like Pacman or Asteroids, the game takes place on one computer screen, but to keep it from having edges and corners the designers made the game "wrap around" -- if anything goes off the right edge of the screen, it comes back at the corresponding spot on the left edge of the screen, and similarly anything that goes off the top of the screen comes back on the bottom of the screen.

A similar model, expanded slightly, is used in many other video games, like early ones in the Final Fantasy series, use essentially the same process to model the surface of a planet. We claim theat any of these videogame words are actually the surface of a torus. Instead of a computer screen, let's put the world on a very flexible flat sheet. To get the left and right edges to match up, we can curl the screen up into a cylinder -- going off one edge takes us around the back of the cylinder. Now, our cylinder has two circular boundaries, coming from the top and bottom of the screen, and to get these to match up we can bend our cylinder up and glue these together to make a torus.

Finally, we can step up a dimension, we generally assume that the three dimensional space we live in is $\mathbb{R}^3$, but what evidence do we have for that? Maybe it's got some different shape, and if we could fly for untold light years in one direction we'd come back to the earth from a different direction! I can recommend Janna Levin's popular science book / memoir "How the Universe got its Spots" for an account of how physicists studied whether patterns in the cosmic microwave background radiation could have been created by the universe being a shape other than $\mathbb{R}^3$.

That discussion may feel out of place; its purpose was to motivate the following definition, which we will then apply to graph theory:

**Definition 4.4.1** An *n-dimensional manifold* is a space that is locally homeomorphic to $\mathbb{R}^2$. A *surface* is a two dimensional manifold. $\Diamond$

### 4.4.2 Graphs on the Torus

The *torus* is another word for the surface of a donut. There are some graphs that can't be on drawn on the sphere, but can be drawn on a torus. But we need a way to represent drawing graphs on the torus just using a normal sheet of paper -- it would be awkward and impractical to hand every student an innertube or a donut at the exam to hand in with their papers.

We will do this by copying the videogame worlds we saw in the introduction. Draw a square to represent the videogame screen, and then draw the graph

inside the square, with the added proviso that if while drawing an edge of the graph we hit the border of the square, we continue the edge at the same point of the opposite side of the square. To make clear what we're doing, it is useful to draw arrows on the edges of the square to indicate how they are being identified -- we put one arrowhead, pointing the same direction, on the left and right edges, to indicate that they are being identified, with the tip end end of one edge being marked to the tip end of the other, and the tail end being matched to the tail end of the other. We draw two arrowheads on the top and bottom edges, also pointing the same direction.

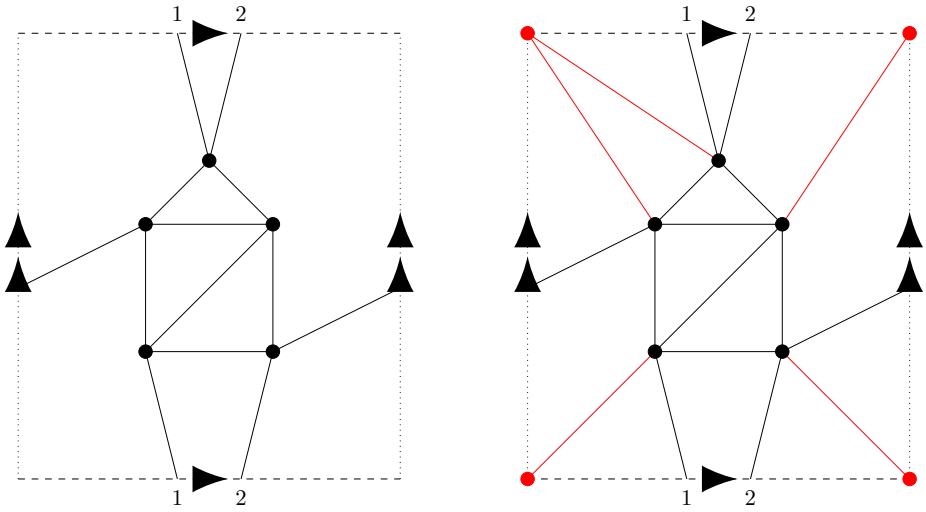



**Figure 4.4.2** $K_5$ and $K_6$ drawn on a torus

This whole process is illustrated in Figure 4.4.2, where the complete graphs on five and six vertices are drawn on the torus. We end with a few short tips and tricks for trying to draw graphs on surfaces.

The first is that it can be complicated to keep track of edges that wrap around the sides of the torus. If lots of them are wrapping around, it can be easy to lose track of which edges are connecting to which others on the opposite side. One way to make this clearer is to label the crossing point on one side letters or numbers in order, and then on the opposite side label with the letters and numbers going in the same direction as indicated by the arrow. Then you know that the edge that crosses at letter $c$, say, on one side, picks up at $c$ on the other edge.

Another idea is to try to minimze the number of such crossings, and draw as much of the graph as possible in the center of the square, and only a few edges wrapping around the torus. One heuristic to follow is to work like we did with the planarity algorithm for Hamiltonian graphs, and try to draw a large cycle as a cycle in the square. Then, connect as many edges and extra vertices as possible through the centre of the cycle, and only when you've ruled that out try to wrap edges around the torus. This is the approach taken in the drawing of $K_5$ in Figure 4.4.2; the outside house is a Hamiltonian cycle, two edges could be drawn in the center of the house, and then the remaining three edges have no choice to be drawn wrapping around the torus.

The drawing of $K_6$ using another trick that is worth explaining. It begins with the drawing of $K_5$ in black, and adds new stuff in red. It appears that four new vertices have been added, one at each corner of the square. But actually, when the edges are identified, the four corners of the sphere get identified into one point on the torus. (Check this by visualizing what happens if you folded

the paper up!). Placing one of the vertices on this graph on the corner can be useful because then edges drawn from this vertex will not need to wrap around the sides of the square.

### 4.4.3 Möbius strip

If you take a piece of paper and roll it up, identifying the two edges in the usual way, one obtains the cylinder. But if instead you identify the edges in the opposite way as usual, you one obtains a different surface called the *Möbius strip*. The Möbius strip is famous for only having one side. In the drawing shown below, if one of the ants walks all the way around the strip, when it returns to where it starts it will be on the opposite side of the strip. Similarly, whereas a cylinder has two boundary cirlces, the Möbius strip only has one.

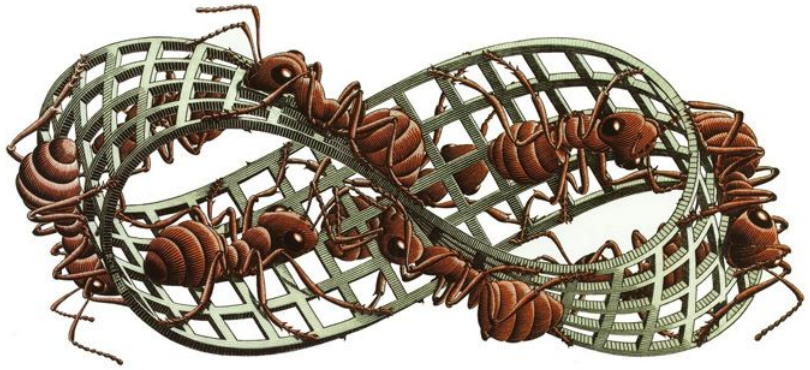


**Figure 4.4.3** M.C. Escher's Möbius Strip II

The fact that the Möbius strip only has one boundary circle has the following surprising consequence, that makes a great "party trick", is to make a Möbius strip by taping up a strip of paper, and the cut it down the very middle of the strip -- you wind up not getting two pieces of paper, but just one!

To actually represent graphs drawn on the Möbius strip, we work similarly to what we did for the torus; we draw a square, and then we draw arrows on the left and right edges to indicate that these edges are drawn together. However, we have one arrow drawn up, and one arrow drawn down, to indicate that the ends of the strip are glued together with a half twist. Since the top and bottom of the Möbius strip do not get identified together at all, we do not draw any arrows on them. Then, if an edge goes off one end the Möbius strip near the top, it comes back on the opposite end near the bottom, and vice versa.

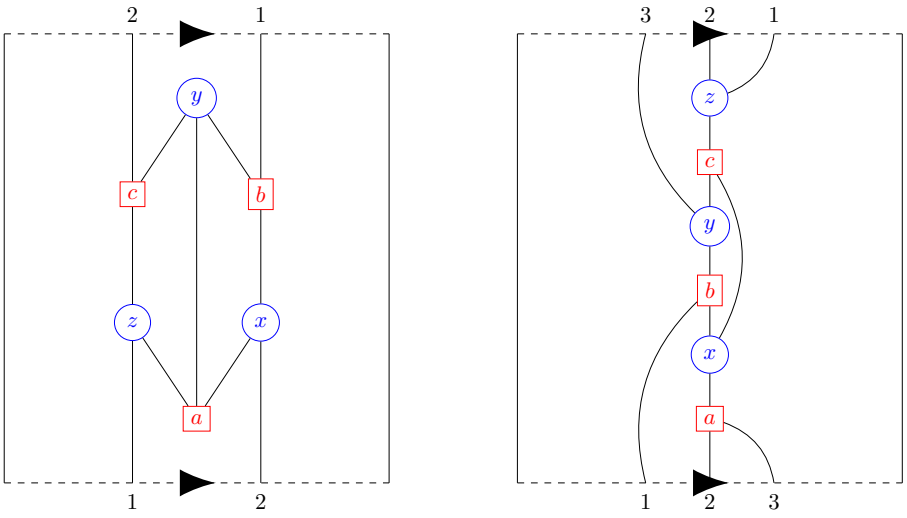



**Figure 4.4.4** Two drawings of $K_{3,3}$ on the Möbius strip

Finally, we explain some heuristics about drawing graphs on surfaces, with reference drawings of $K_{3,3}$ on the Möbius strip. As discussed at the end of the section of the torus, it can be useful to follow part of the reasoning used in the planarity algorithm for Hamiltonian cycles when trying to draw graphs on surfaces other than the sphere. That is, start with a large cycle in the graph -- for $K_{3,3}$, with $a, b, c$ red vertices, and $x, y, z$ the blue vertices, we will use the cycle $axbycza$. It may be that this is cycle is drawn in the plane as a circle, as the in the left hand example; then we try to connect as many edges as possible through the centre of the circle, and then do the rest on the outside, possible wrapping around the Möbius strip.

But the Jordan Curve Theorem (that a circle has an inside and an outside) only holds for the sphere -- on other surfaces, there are always curves where this isn't true. Whether of necessity, or choice, it might be that our large cycle is drawn as one of these loops that doesn't have an inside or an outside. On the right hand side of Figure 4.4.4, we have drawn the cycle $axbycz$ as the centre of the Möbius strip.

Finally, we note that in general for a surface, there are multiple different ways to draw curves; in Figure 4.4.5, we have drawn our chosen Hamiltonian cycle as a curve that wraps *twice* about the Möbius strip. What happens if you cut a physical Möbius strip along this line?

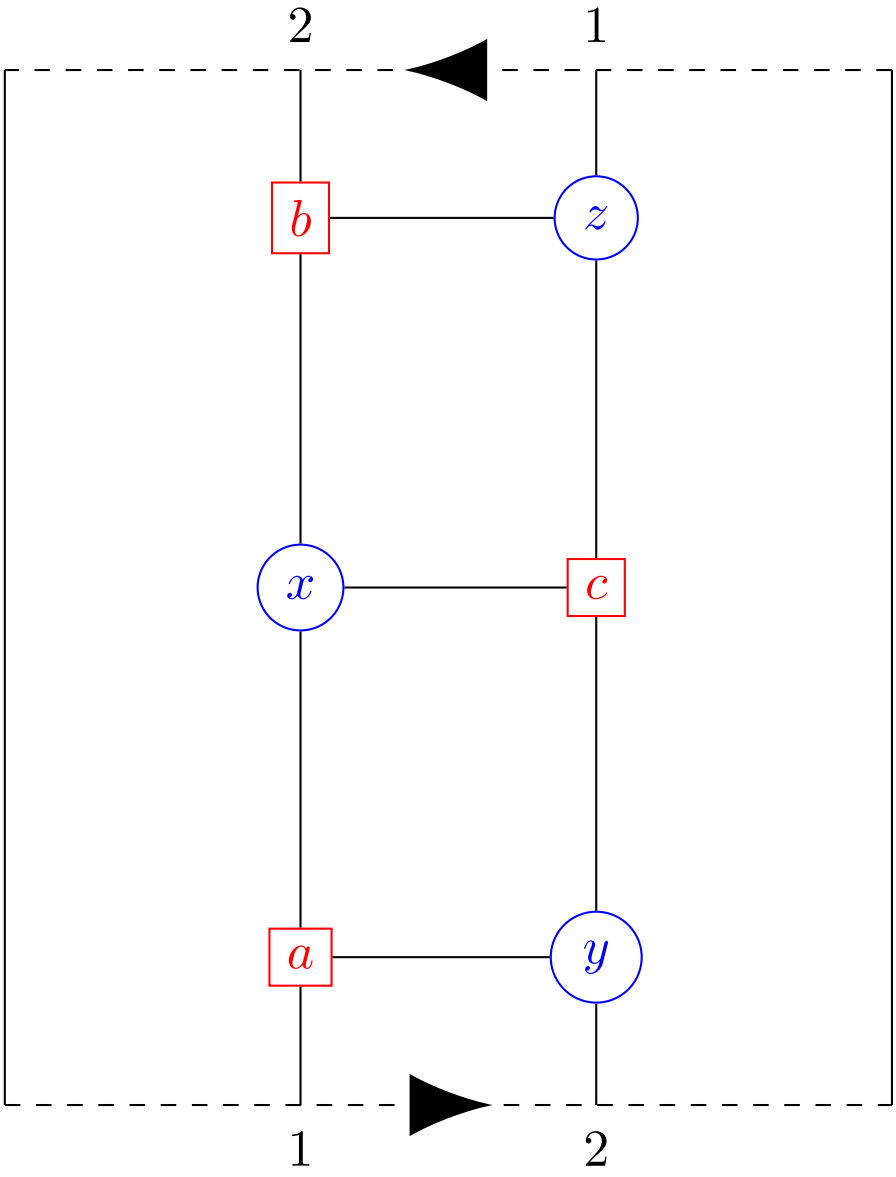



**Figure 4.4.5** A third drawing of $K_{3,3}$ on the Möbius strip

## 4.5 Euler's Theorem

This section cover's Euler's theorem on planar graphs and its applications. After defining faces, we state Euler's Theorem by induction, and gave several applications of the theorem itself: more proofs that $K_{3,3}$ and $K_5$ aren't planar, that footballs have five pentagons, and a proof that our video game designers couldn't have made their map into a sphere without doing something very strange.

### 4.5.1 Counting faces

A **face** of a planar drawing of a graph is a region bounded by edges and vertices and not containing any other vertices or edges.

Figure 4.5.1 shows a planar drawing of a two graphs. The left graph has determines 5 regions, since we also count the unbounded region that surrounds the drawing.

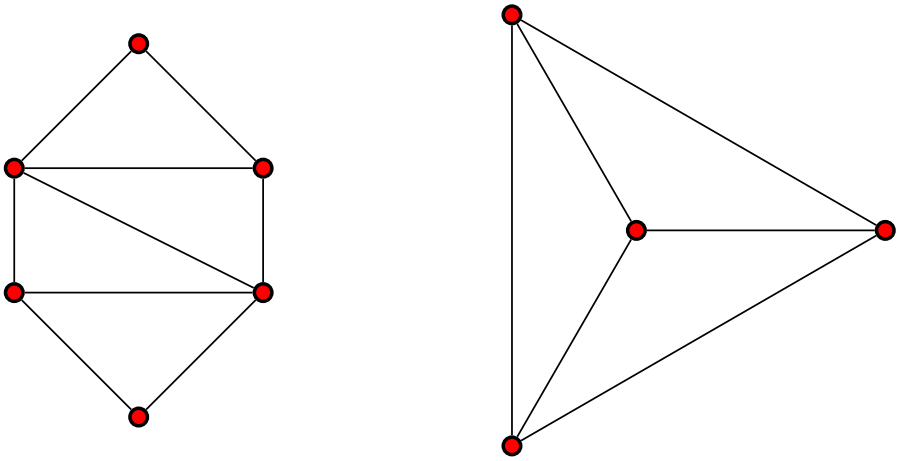


**Figure 4.5.1** Two planar graphs

What happens if we compute the number of vertices minus the number of edges plus the number of faces for these drawings? We have

$$6 - 9 + 5 = 2$$
$$4 - 6 + 4 = 2$$

While it might seem like a coincidence that this computation results in 2 for these planar drawings, there's a more general principle at work here, and in fact it holds for *any* planar drawing of *any* planar graph.

In fact, the number 2 here actually results from a fundamental property of the plane, and there are a corresponding theorems for other surfaces. However, we only need the result as stated above.

**Theorem 4.5.2 Euler's Formula.** *Let* $\mathbf{G}$ *be a connected planar graph with* $n$ *vertices and* $m$ *edges. Every planar drawing of* $\mathbf{G}$ *has* $f$ *faces, where* $f$ *satisfies*

$$n - m + f = 2.$$

*Proof.* Our proof is by induction on the number $m$ of edges. If $m = 0$, then since $\mathbf{G}$ is connected, our graph has a single vertex, and so there is one face. Thus $n - m + f = 1 - 0 + 1 = 2$ as needed. Now suppose that we have proven Euler's formula for all graphs with less than $m$ edges and let $\mathbf{G}$ have $m$ edges. Pick an edge $e$ of $\mathbf{G}$. What happens if we form a new graph $\mathbf{G}'$ by deleting $e$ from $\mathbf{G}$? If $\mathbf{G}'$ is connected, our inductive hypothesis applies. Say that $\mathbf{G}'$ has $n'$ vertices, $m'$ edges, and $f'$ faces. Then by induction, these numbers satisfy

$$n' - m' + f' = 2.$$

Since we only deleted one edge, $n' = n$ and $m' = m - 1$. What did the removal of $e$ do to the number of faces? In $\mathbf{G}'$ there's a new face that was formerly two faces divided by $e$ in $\mathbf{G}$. Thus, $f' = f - 1$. Substituting these into $n' - m' + f' = 2$, we have

$$n - (m - 1) + (f - 1) = 2 \iff n - m + f = 2.$$

Thus, if $\mathbf{G}'$ is connected, we are done. If $\mathbf{G}'$ is disconnected, however, we cannot apply the inductive assumption to $\mathbf{G}'$ directly. Fortunately, since we removed only one edge, $\mathbf{G}'$ has two components, which we can view as two connected

graphs $\mathbf{G}_1'$ and $\mathbf{G}_2'$. Each of these has fewer than $m$ edges, so we may apply the inductive hypothesis to them. For $i = 1, 2$, let $n_i'$ be the number of vertices of $\mathbf{G}_i'$, $m_i'$ the number of edges of $\mathbf{G}_i'$, and $f_i'$ the number of faces of $\mathbf{G}_i'$. Then by induction we have

$$n_1' - m_1' + f_1' = 2 \quad \text{and} \quad n_2' - m_2' + f_2' = 2.$$

Adding these together, we have

$$(n_1' + n_2') - (m_1' + m_2') + (f_1' + f_2') = 4.$$

But now $n = n_1' + n_2'$, and $m_1' + m_2' = m - 1$, so the equality becomes

$$n - (m-1) + (f_1' + f_2') = 4 \iff n - m + (f_1' + f_2') = 3.$$

The only thing we have yet to figure out is how $f_1' + f_2'$ relates to $f$, and we have to hope that it will allow us to knock the 3 down to a 2. Every face of $\mathbf{G}_1'$ and $\mathbf{G}_2'$ is a face of $\mathbf{G}$, since the fact that removing $e$ disconnects $\mathbf{G}$ means that $e$ must be part of the boundary of the unbounded face. Further, the unbounded face is counted twice in the sum $f_1' + f_2'$, so $f = f_1' + f_2' - 1$. This gives exactly what we need to complete the proof. ■

**Remark 4.5.3 Alternative method of dealing with the second case.** In our proof of Euler's theorem, the most complicated part was dealing with the situation if the edge $e$ disconnects our graph $\mathbf{G}$ when we remove it. In this case, instead of deleting the edge $e$ we can contract it -- that is, shrink it to a point. This would have result in a graph that is still planar and still connected, but with one less edge ($e$ is no longer around), and one less vertex (the two vertices $e$ connects are now merged into one). The number of faces remains unchanged. So the number of edges and the number of faces each decreased by one, these two changes cancel out when we calculate $n - m + f$, and hence both are equal to two.

### 4.5.2 Applications of Euler's theorem

By itself, Euler's theorem doesn't seem that useful: there are three variables (the numbers of edges, vertices, and faces) and only one equation between them, so there are still lots of degrees of freedom. For it to be particularly useful, we want to have other relationships between these numbers. In many applications, these relationships can come from handshaking.

Recall that Euler's handshaking lemma said that

$$\sum_{v \in G} d(v) = 2|E(G)|,$$

the sum of the degrees of all the vertices is twice the number of edges. If we had some knowledge about the degrees of these vertices, we could get another relationship between the number of vertices and the number of edges. For example, if $G$ is regular of degree $k$, then every vertex has degree $k$, and hence the sum of all the degrees is just $kn$. Hence, handshaking would tell us that $kn = 2m$, and we would have another relationship between the three variables $m, n$ and $f$.

Similarly, there is a handshaking between faces and edges. Let the *degree* of a face be the number of edges that occur around it -- so, a triangle would have degree three. Then, if we sum up the degrees of all the faces, we're counting each edge twice again -- once from the face on its left, and once from the face

on its right. so we have

$$\sum_{f \in \text{faces}(G)} d(f) = 2|E(G)|$$

Note that this is just the usual vertex-edge handshaking for the dual graph.

Thus, vertex-edge and face-edge handshaking can potentially give us two other sources of relationships between the numbers of vertices, edges, and faces. Most applications of Euler's theorem proceed by combining all three relationships, as we shall see.

**Lemma 4.5.4** *$K_5$ isn't planar*

*Proof.* We give a proof by contradiction. Suppose $K_5$ was planar, and draw it on the plane. We know that $K_5$ has 5 vertices and $\binom{5}{2} = 10$ edges, and so by Euler theorem we know that any drawing of it must have 7 faces. We now use edge-face handshaking to get a contradiction.

What could the degrees of the faces be? We don't know for sure, but we know that none of the faces could have degree one or two, as then the edges would form a loop or multiple edges between two vertices, but $K_5$ is simple. Hence, every face must have $d(f) \geq 3$. But then handshaking gives:

$$20 = 2e = \sum d(f) \geq \sum 3 = 3 \cdot 7 = 21$$

which is the desired contradiction, and so we conclude that $K_5$ is not, in fact, planar. ■

It is a good exercise to adapt this proof to prove that $K_{3,3}$ isn't planar; one needs to use the extra fact that $K_{3,3}$ doesn't have any three cycles (why not?)

We now prove that footballs have 12 pentagons. More precisely, use the shorthand *football graph* to mean any trivalent graph drawn on the plane so that every face is a pentagon or hexagon. Then we have:

**Theorem 4.5.5 The football theorem.** *Let* **G** *be a football grpah drawn on the plane, with $P$ pentagonal faces, and $H$ hexagonal faces. Then $P = 12$.*

*Proof.* Let $V, E, F$ denote the number of vertices, edges, and faces of **G**. Since every face is a hexagon or pentagon, we have $E = P + H$, and substituting this into Euler's theorem gives:

$$V - E + P + H = 2 \tag{4.5.1}$$

Now we turn handshaking. Since **G** is trivalent, every vertex has degree three, and so vertex-edge handshaking becomes:

$$2E = 3V \tag{4.5.2}$$

Finally, since there are $P$ pentagonal faces and $H$ hexagonal faces, face-edge handshaking becomes:

$$2E = 5P + 6H \tag{4.5.3}$$

Multiplying (4.5.1) by six gives:

$$6V - 6E + 6P + 6H = 12$$

Multiplying (4.5.2) by two gives $6V - 4E = 0$, which we can use to eliminate $V$, giving

$$-2E + 5P + 6H = 12$$

Finally, using (4.5.3) we can eliminate both $E$ and $H$ in one go, being left with $P = 12$ as desired. ■

Finally, we prove that given some sensible restraints, video game designers cannot make a world map that's a sphere. A videogame world locally looks like a square grid -- with every vertex and face having degree four.

**Theorem 4.5.6 The videogame theorem.** *It is impossible to draw for a graph to be drawn on the sphere so that every vertex and every face has degree 4.*

*Proof.* Since every vertex has degree 4, vertex-edge handshaking gives $2E = 4V$, and since every face has degree 4, face-edge handshaking gives $2E = 4F$. Thus, we see $V = F = E/2$, and plugging this in gives:

$$V - E + F = E/2 - E + E/2 = 0$$

which contradicts Euler's Theorem. Hence, such a graph on a sphere is not possible. ■

## 4.6 Exercises

**1.** Show that the Petersen graph has a cycle that uses all the vertices but one. Give another proof that the Petersen graph is not planar by modifying the planarity algorithm for Hamiltonian graphs to deal with this extra vertex.

**2.** Draw $K_{4,4}$ and $K_7$ on the torus.

**3.** Use the planarity algorithm to show that the given graph **G** is planar, and draw a plane graph isomorphic to it. Explain how you might obtain a non-planar graph by adding one extra edge, and for your non-planar graph, find a subgraph that's a subdivision of $K_5$ or $K_{3,3}$.

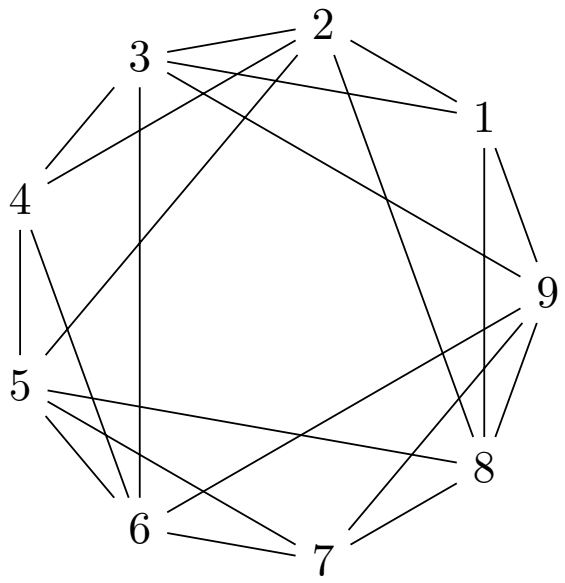



**Figure 4.6.1** A Hamiltonian graph **G**

**4.**

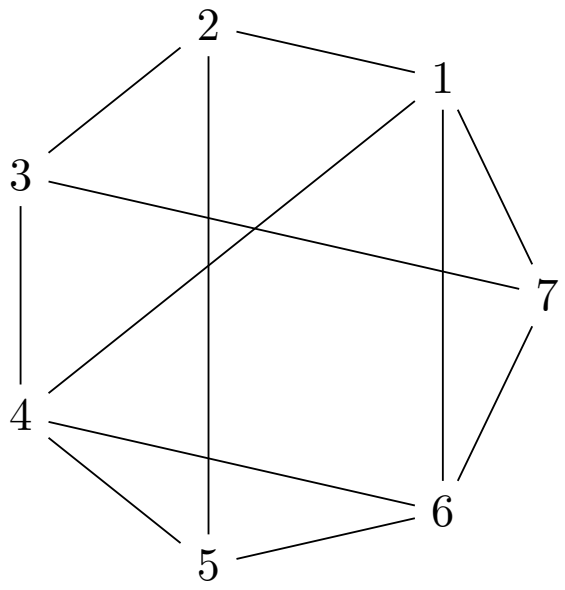



**Figure 4.6.2** A Hamiltonian graph **H**

Let **H** be the graph shown in Figure 4.6.2. Show that **H** is not planar in two ways: using the planarity algorithm, and using Kuratowski's theorem.

Draw **H** on the Möbius strip and on a torus without the edges crossing.

**5.** A connected plane graph has faces of degrees 3 and 10 only, and every vertex has degree at least 3. Prove that it must have fewer faces of degree 10 than of degree 3. If every vertex has degree 3, prove that the number of faces of degree 10 must be a multiple of 3 and the number of faces of degree 3 must be a multiple of 4.

As an example, how many faces of degree 3 and degree 10 does the truncated dodecahedron possess? (A truncated dodecahedron is obtained by sliving off each vertex of a dodecahedron to give a triangle.)

# Chapter 5

# Colourings

This chapter covers several types of colouring questions on graphs. The initial motivation for these questions comes from an early question about colouring the countries on maps so that adjacent countries have different colours.

## 5.1 Chromatic number

The study of graph colourings began with the colouring of maps. Usually on a map, different regions (countries, counties, states, etc.) are visually distinguished from each other by giving each one a different colour, with the idea that adjance regions should have different colours so that boundaries can be easily seen. For instance, in this old road map of England and Wales, each county is coloured either red, yellow blue or green, and bordering counties have different colours.


**Figure 5.1.1** A historical example of a map colouring. Image courtesy Cartography Associates under a creative commons license

Note that in the map above, only four colours are used. In 1852 Francis Guthrie suggested that for any possible map drawn on a piece of a paper, four colours would be enough. Guthrie's conjecture wasn't proven for more than a hundred years later.

Let's make Guthrie's conjecture precise and connect it to graph theory. Note that as in Example , it makes sense to work with essential the dual picture

to the map; we make the regions of the map into vertices, and we put an edge between two regions if they share a vertex. Then we are lead to the following definitions.

**Definition 5.1.2 Colourings and Chromatic number.** Let $\mathbf{G}$ be a graph. A *$k$-colouring* (or sometimes *vertex colouring*) of $\mathbf{G}$ with $k$ colours is an assignment of one of $k$ colours to each of the vertices of $\mathbf{G}$ so that adjacent vertices have different colours.

More formally, a $k$-colouring is a function $f : V(\mathbf{G}) \to \{1, \ldots, k\}$ so that if $v \sim w$ than we have $f(v) \neq f(w)$.

The *chromatic number* of a graph $\mathbf{G}$, written $\chi(\mathbf{G})$, is the least number of colours needed to colour the vertices of $G$ so that adjacent vertices are given different colours; that is, it's the least $k$ so that there exists a $k$-colouring of $\mathbf{G}$. ◊

The most basic problem you will have to complete about these is the following: given a graph $\mathbf{G}$, determine its chromatic number$\chi(\mathbf{G})$. Because the chromatic number is the *least* number of colours with which it is possible to colour $\mathbf{G}$, showing that $\chi(\mathbf{G}) = N$ will always require two steps:

1. Show that $\mathbf{G}$ admits a colouring with $\chi(\mathbf{G}) = N - 1$ colours
2. Show that $\mathbf{G}$ does not admit a colouring with fewer colours.

**Example 5.1.3 Complete graph.** What's the chromatic number $\chi(K_n)$ of the complete graph? Since every vertex is adjacent to every other vertex, any two vertices need to have different colours, and so $\chi(K_n) \geq n$. But certainly if we colour every vertex a different colour, then two adjacent vertices have the same colour, and that's a valid colouring of $K_n$, so $\chi(K_n) \leq n$. So $\chi(K_n) = n$. □

**Example 5.1.4 Trees.** Suppose that $T_n$ is a tree on $n \geq 2$ vertices. Then $T_n$ has an edge, and the two vertices on this edge must be different colours, and so $\chi(T_n) \geq 2$. On the other hand, we can colour any tree with two colours as follows: pick any vertex, and colour it blue; then pick any vertex next to it and colour it red, then we can colour the vertices next to that blue, and colour the vertices next to those red, and continuing on outwards from our starting vertex. Hence, $\chi(T_n) \leq 2$ and so $\chi(T_n) = 2$.

Another way of phrasing this is that along any path we colour the vertices alternating red-blue-red-blue-red-blue. This wouldn't work for a general graph, because there may be two paths of different lengths between a pair of vertices $v$ and $w$. But in trees because there is always exactly one path between any two vertices, and so once we colour one vertex, there's a unique way to colour all the others with two colours in this way. □

**Example 5.1.5 Cyclic graphs $C_n$.** As with trees, as long as $n \geq 2$ the graph has at least one edge, and thus has $\chi(C_n) \geq 2$. Can we colour $C_n$ with two colours?

If we could, the vertices would have to alternate red-blue-red-blue all the way around. This works if $n$ is even, but if $n$ is odd then the vertex we started with would have the same colour as the vertex we ended with, but they're adjacent. Thus, when $n$ is odd we need at least three colours to colour the graph, but it's easy to do with three colours -- we can alternate red-blue-red-blue, but make the very last vertex green, for instance.

Summarizing, we have:

$$\chi(C_n) = \begin{cases} 2 & n \text{ odd} \\ 3 & n \text{ even} \end{cases}$$

□

In fact, as thinking about the examples of trees and cycles should show, we've already met one instance of the chromatic number -- a graph $\mathbf{G}$ is bipartite if and only if $\chi(\mathbf{G}) = 2$, as follows immediately from the definition.

Another useful observation about the examples we've seen is that, since the chromatic number $\chi(\mathbf{G})$ is the *least* number of colours needed to colour $\chi(\mathbf{G})$, to show that $\chi(\mathbf{G}) = s$ requires doing two things:

1. Showing that $\mathbf{G}$ *can* be coloured with $s$ colours, and hence $\mathbf{G} \leq s$
2. Showing that $\mathbf{G}$ *can't* be coloured with $s-1$ colours, and hence $\mathbf{G} \geq s$

**Example 5.1.6 The Wheel graph.** The wheel graph $W_n$ consists of an $n$-cycle together with one additional vertex, that is connected to all vertices of the $n$-cycle. Note that this with this convention, $W_n$ confusingly has $n-1$ vertices; other people may use a different convention where $W_n$ has $n$ vertices, but then it only has a $n-1$ vertices on the actual wheel.

Since the central vertex is connected to all other vertices, once we colour it, we can never use that colour again. But deleting that vertex we just have the $n$-cycle, and we already know the chromatic number of that. So we have $\chi(W_n) = \chi(C_n) + 1$. □

**Definition 5.1.7** We write $\Delta(\mathbf{G})$ for the maxium degree of any vertex in $\mathbf{G}$:

$$\Delta(\mathbf{G}) = \max_{v \in \mathbf{G}} d(v)$$

◊

**Theorem 5.1.8** *We have* $\chi(\mathbf{G}) \leq \Delta(\mathbf{G}) + 1$

*Proof.* We need to show that we can colour any graph $\mathbf{G}$ with $\Delta(G) + 1$ colours. But we can just colour the vertices of $\mathbf{G}$ one by one in whatever order we want. When we go to colour the $i$th vertex $v_i$, we look at the $d(v_i)$ vertices adjacent to $v_i$. Some of them may not be coloured yet, in which case they don't affect anything, but for each vertex adjacent to $v_i$ that is coloured, we can't use that colour for $v_i$.

So there are most $d(v_i) \leq \Delta(\mathbf{G})$ colours we have to avoid; if we have $\Delta(\mathbf{G})+1$ colours to choose from we can always find one that hasn't been used at a vertex adjacent to $v_i$. ■

## 5.2 Chromatic index and applications

It is a natural twist of the definition of chromatic number to try to colour the edges of a graph instead; the least number of colours needed is the called the *chromatic index*. After introducing this concept and giving some examples, we give some story problem type questions that boil down to finding either the chromatic number or chromatic index.

**Definition 5.2.1 Chromatic index.** The *chromatic index* $\chi'(\mathbf{G})$ of a graph $\mathbf{G}$ is the least number of colours needed to colour the edges of $\mathbf{G}$ so that any two edges that share a vertex have different colours. ◊

Not that as with the chromatic number, since the chromatic index $\chi'(\mathbf{G})$ is the minimum number of colours such that the edges can be coloured with adjacent edges having different colours, to show $\chi'(\mathbf{G}) = N$ typically requires two steps:

1. Prove that the edges of $\mathbf{G}$ can be coloured with $N$ colours

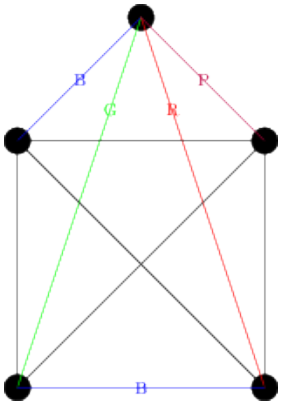
2. Prove that the edges of **G** cannot be coloured with less than $N$ colours

**Example 5.2.2 The complete graph $K_4$.** Let's find $\chi'(K_4)$. Picking any vertex $v$, there are three edges incident to $v$, and none of these edges can have the same colour (as they all meet at $v$). Hence, we have $\chi'(K_4) \geq 3$.

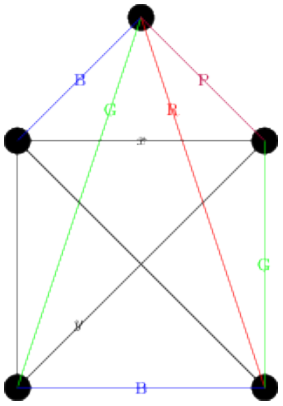
On the other hand, it is easy to colour the edges of $K_4$ with three colours, as seen below, and so $\chi'(K_4) \leq 3$, and hence $\chi'(K_4) = 3$. □

**Example 5.2.3 The complete graph $K_5$.** Now, let's move on to $K_5$. Again, looking at any vertex we see all the edges adjacent to that vertex must be different colours, and so we have $\chi'(K_5) \geq 4$. Let's try to colour the edges of $K_5$ with 4 colours.

Suppose we coloured the four edges adjacent to the top vertex blue, green, red and purple, from left to right, and now look at the bottom edge. It is adjacent to edges coloured green and red, and so must be blue or purple. By symmetry, it's equivalent to colour it either colour, so let's suppose it's blue, giving us the following picture:

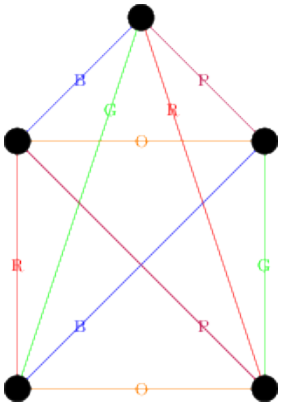



Now the edge on the right is adjacent to edges coloured red, blue and purple, and so must be green. But now we have a problem -- consider the edges labeled $x$ and $y$ in the next drawing:




Both edges share vertices with edges coloured green, blue, and purple, and hence each would need to be coloured red. But they also share a vertex with each other, and so cannot both be coloured red. So we see $\chi'(K_5) \geq 5$.

On the other hand, it is easy to colour the edges of $K_5$ with 5 colours: colour each edge in the outside pentagon a different colour. For each edge in the outside pentagon there will be a unique edge in the inside star that does meet that edge (the one "parallel" to it) -- draw that edge the same colour. That results in the following colouring:




□

In the examples above, we found lower bounds for $\chi'(\mathbf{G})$ by considering the degrees of vertices; this argument easily adapts in general.

**Theorem 5.2.4** *For any graph* $\mathbf{G}$ *we have* $\chi'(\mathbf{G}) \geq \Delta(\mathbf{G})$

*Proof.* Let $v \in \mathbf{G}$ be a vertex of maximal degree $d(v) = \Delta(\mathbf{G})$. Then none of the $\Delta(\mathbf{G})$ edges incident to $v$ can be coloured the same colour, and so we have $\chi'(\mathbf{G}) \geq \Delta(\mathbf{G})$ ■

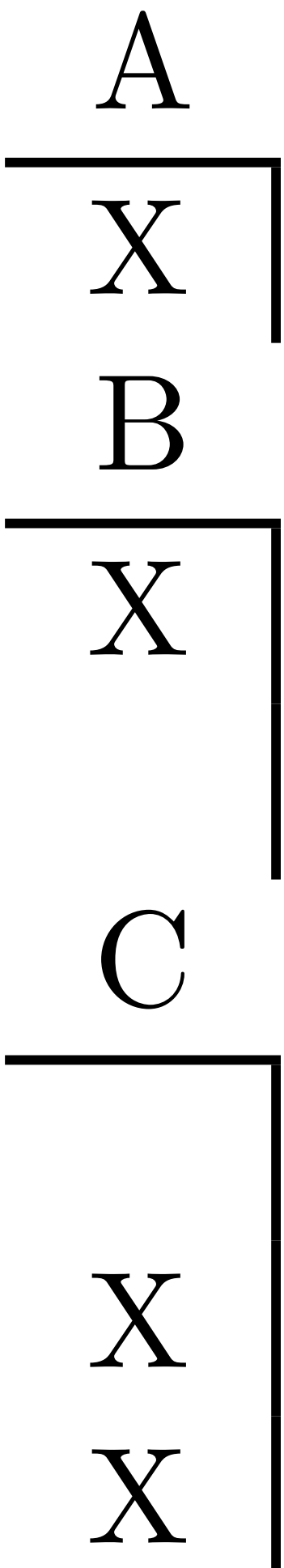
It turns out that this nearly determines the chromatic index $\chi'(\mathbf{G})$ -- it can be at most one more than $\Delta(\mathbf{G})$:

**Theorem 5.2.5 Vizing's Theorem.** *For any graph* $\mathbf{G}$ *we have* $\Delta(\mathbf{G}) \leq \chi'(\mathbf{G}) \leq \Delta(\mathbf{G}) + 1$

*Proof.* The lower bound was just proved in the previous theorem. The other direction is more difficult. ■

We now show how determining the chromatic number and chromatic index can show up as part of story questions.

Suppose there are six friends, Alice, Bob, Charlie, Dora, Elizabeth and Frank, and there is the following graph between then:


A
X
B
X
C
X
X

Here are two word problems related to **G**:

1. The friends want to divide into groups, but the edges indicate people who currently annoy each other. What's the least number of groups the friends can divide into groups so that no group contains two people who annoy each other?

2. The friends want to hold a snooker tournament, with everyone playing three matches; the edges indicate pairs of friends who will play against each other. If multiple matches can be played each day, but each person can only be involved in one match a day, how many days are necessary to hold the tournament?

The first case concerns the chromatic number -- each group of people will be the people who have the same colour, and we don't want vertices with an edge between them to have the same colour.

The second case concerns the chromatic index -- the edges are the games that are being played, and all edges that are the same colour will be played on the same day.

Let us quickly compute the chromatic number and chromatic index of the graph **G** above. To compute the chromatic number, we observe that the graph contains a triangle, and so the chromatic number is at least 3. But it is easy to colour the vertices with three colours -- for instance, colour A and D red, colour C and F blue, and colur E and B green. So $\chi(\mathbf{G}) = 3$.

To compute $\chi'(G)$, since $A$ has degree three we have $\chi'(\mathbf{G}) \geq 3$. On the other hand, it is easy to colour the edges with three colours -- for instance, colour AB, CE and DF red, colour AE, CD and BF blue, and colour AC, BD and EF green. So $\chi'(\mathbf{G}) = 3$ as well.

## 5.3 Introduction to the chromatic polynomial

For the chromatic number, we were asking whether or not it was possible to colour the vertices of **G** with a given number of colours. The chromatic polynomial extends this question, and asks the following. Suppose we have $k$ colours. How many different ways can we colour the vertices of **G**? It is *not* clear that this should be a polynomial in $k$, and proving that it is in fact a polynomial in $k$ is the highlight of the section.

### 5.3.1 Definition and examples

**Definition 5.3.1 The chromatic polynomial $P_{\mathbf{G}}$.** The chromatic polynomial $P_{\mathbf{G}}$ of a graph **G** is the function that takes in a non-negative integer $k$ and returns the number of ways to colour the vertices of **G** with $k$ colours so that adjacent vertices have different colours. ◊

It is immediate from the definition of the chromatic polynomial that $\chi(\mathbf{G})$ is the least positive number $k$ such that $\chi_{\mathbf{G}}(k) \neq 0$.

It is *not* immediate from the definition of the chromatic polynomial that it is, in fact, a polynomial. In fact, proving that will take a little bit of work, and we postpone this until the end. First, we look at some examples of the chromatic polynomial; in many cases it is possible to easily compute the chromatic polynomial by working "vertex by vertex".

**Example 5.3.2 The empty graph $E_n$.** Recall that the empty graph $E_n$ has $n$ vertices and no edges. To compute $P_{E_n}(k)$ we need to count the number of ways to colour the vertices with $k$ colours. But since $E_n$ has no edges, we can

colour each of the $n$ vertices any of the $k$ colours; the choices are completely independent. So $P_{E_n}(k) = k^n$ □

**Example 5.3.3 The complete graph $K_n$.** Let's label the vertices $v_0, \ldots, v_{n-1}$, and colour them one by one in the given order. When we colour the first vertex $v_0$, no other vertices have been coloured, and we can use whichever of the $k$ vertice we like. However, when we go to colour $v_1$ we note that it is adjacent to $v_0$, and so whatever colour we used for $v_0$ we can't for $v_1$, and so we have $k-1$ colours to choose for $v_1$

Continuing in this way, we see that since all the vertices are adjacent, they all most have different colours. So when we go to colour $v_i$, we have already coloured $v_o, \ldots, v_{i-1}$ with $i$ different colours, and we can't use any of these to colour $v_i$, and so we have $k-i$ choices to colour $v_i$.

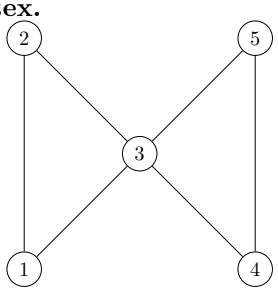
Putting it all together, we see that:

$$P_{K_n}(k) = k \cdot (k-1) \cdot (k-2) \cdots k - n + 1$$

□

**Example 5.3.4 Two triangles joined at a vertex.**

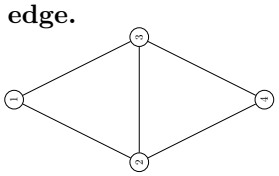
Consider the graph $\mathbf{G}$ consisting of two triangles joined at the right a vertex, shown at the right. We can calulate $P_{\mathbf{G}}(k)$ by working vertex by vertex: there are $k$ ways to colour vertex 1, but then when we colour vertex 2 it can't be the same as vertex 1, and so there are $k-1$ ways to colour it. Vertex three is adjacent to both one and two, so there are $k-2$ ways to colour it, and vertex 4 is adjacent to one coloured vertex, vertex 3, so there are again $k-1$ ways to colour it, and finally vertex 5 is adjacent to vertices 3 and 4 and so we have $k-2$ ways to colour it.Putting it together, we see $P_{\mathbf{G}}(k) = k(k-1)^2(k-2)^2$.




□

**Example 5.3.5 Two triangles joined along an edge.**

Consider the graph $H$ consisting of two triangles joined along an edge. shown at the right. We can calulate $P_H(k)$ by working vertex by vertex: there are $k$ ways to colour vertex 1, but then when we colour vertex 2 it can't be the same as vertex 1, and so there are $k-1$ ways to colour it. Vertex three is adjacent to both one and two, so there are $k-2$ ways to colour it, and vertex 4 is adjacent to vertices 2 and 3, which have different colours as they are adjacent to each other, so there are $k-2$ ways to colour it. Putting it together, we see $P_H(k) = k(k-1)(k-2)^2$



□

## 5.3.2 Gluing

What can we say about the chromatic polynomial of a graph $\mathbf{G}$ that's the disjoint union of two smaller graphs: $\mathbf{G} = \mathbf{G}_1 \sqcup \mathbf{G}_2$?

We already covered the extreme case where $\mathbf{G} = E_n$ is just a disjoint union of vertices; we could colour each vertex independently of the others, as there

were no edges between them at all. A similar argument works in general to give the following.

**Proposition 5.3.6** *Let* $\mathbf{G} = \mathbf{G}_1 \sqcup \mathbf{G}_2$ *be a disconnected graph. Then* $P_{\mathbf{G}}(k) = P_{\mathbf{G}_1}(k)P_{\mathbf{G}_2}(k)$

*Proof.* A colouring of $\mathbf{G}$ with $k$ colours gives a colouring of $\mathbf{G}_1$ with $k$ colours and a colouring of $\mathbf{G}_2$ with $k$ colourings. Similarly, since $\mathbf{G}_1, \mathbf{G}_2$ are disconnected, how we colour one will have no effect on what colourings are possible for the other. Hence, colouring $\mathbf{G}$ is exactly the same as colouring $\mathbf{G}_1$ and $\mathbf{G}_2$ ■

It gets more interesting if $\mathbf{G}$ is *almost* the disjoint union of two graphs, by which we mean that $\mathbf{G}$ is the union of two subgraphs $\mathbf{G}_1$ and $\mathbf{G}_2$ that are nearly disjoint -- perhaps they share a single vertex, or two vertices connected by an edge. In these cases, there are nice "gluing" formulas relating the chromatic polynomials of $\mathbf{G}, \mathbf{G}_1$ and $\mathbf{G}_2$; as $\mathbf{G}_1 \cap \mathbf{G}_2$ grows more complicated it is still possible to say something but the answer gets more complicated and not worth it.

**Proposition 5.3.7** *Suppose* $\mathbf{G}$ *has two subgraphs* $\mathbf{G}_1$ *and* $\mathbf{G}_2$*, so that their union is all of* $\mathbf{G}$*, but their intersection is a single vertex* $v$*, i.e.* $\mathbf{G}_1 \cup \mathbf{G}_2 = \mathbf{G}$ *and* $\mathbf{G}_1 \cap \mathbf{G}_2 = \{v\}$*. Then we have the following relation between their chromatic polynomials:*

$$P_{\mathbf{G}}(k) = \frac{1}{k} P_{\mathbf{G}_1}(k) P_{\mathbf{G}_2}(k)$$

*Proof.* As in the proof of colourings of disjoint unions, a colouring of $\mathbf{G}$ gives a colouring of both $\mathbf{G}_1$ and $\mathbf{G}_2$ by restriction, but we don't get any two colourings: both $\mathbf{G}_1$ and $\mathbf{G}_2$ contain $v$, and our two colourings must both make $v$ the same colouring.

In the other direction, if we have colourings of $\mathbf{G}_1$ and $\mathbf{G}_2$ that have the same colour at $v$, it is clear that we can glue them together to get a colouring of $\mathbf{G}$. So the question reduces to the following: given that we want vertex $v$ to have a fixed colour, how many colourings of $\mathbf{G}_2$ are there that colour $v$ this colour?

The $k$ colours are completely interchangeable, however; we could just change the names of each one. Thus, it is clear that there are as many colourings of $\mathbf{G}_2$ where $v$ is red as there are where $v$ is blue as there are where $v$ is any given colour. Hence, if we have $k$ colours to use, exactly $1/k$ of all colourings of $\mathbf{G}_2$ will ahve $v$ any given colour, namely $P_{\mathbf{G}_2}(k)/k$

Thus, to colour $\mathbf{G}$ with $k$ colours, we first could first colour $\mathbf{G}_1$ in one of the $P_{\mathbf{G}_1}(k)$ ways. This will give us a some fixed colour of $v$, and we saw above that there are $P_{\mathbf{G}_2}(k)/k$ colourings of $\mathbf{G}_2$ where $v$ has this colour, and so we have the result. ■

**Example 5.3.8 Two triangles joined at a vertex, again.** As an example, we revisit our previous example. For $\mathbf{G}$, two triangles joined at a vertex, we have $\mathbf{G}_1 \cong \mathbf{G}_2 \cong C_3$. Since $C_3 \cong K_3$, we know $P_{C_3}(k) = k(k-1)(k-2)$. Thus, we have:

$$P_{\mathbf{G}}(k) = \frac{P_{C_3}(k)P_{C_3}(k)}{k} = \frac{k^2(k-1)^2(k-2)^2}{k} = k(k-1)^2(k-2)^2$$

□

**Proposition 5.3.9** *Suppose* $\mathbf{G}$ *has two subgraphs* $\mathbf{G}_1$ *and* $\mathbf{G}_2$*, so that their union is all of* $\mathbf{G}$*, but their intersection is a single vertex edge* $e$ *connecting two vertices* $v$ *and* $u$*, i.e.* $\mathbf{G}_1 \cup \mathbf{G}_2 = \mathbf{G}$ *and* $\mathbf{G}_1 \cap \mathbf{G}_2 = \{e\}$*. Then we have the*

*following relation between their chromatic polynomials:*

$$P_{\mathbf{G}}(k) = \frac{1}{k(k-1)} P_{\mathbf{G}_1}(k) P_{\mathbf{G}_2}(k)$$

*Proof.* The proof is extremely similar to that of the previous proposition. A colouring of $\mathbf{G}$ gives us colourings of $\mathbf{G}_1$ and $\mathbf{G}_2$, but not any two colourings: they need to match at both $v$ and at $u$.

Now, $v$ could be any of the $k$ colours, but $u$, being adjacent to $v$, can't be the same colour, and so it has $k-1$ possibilities given a choose of colour for $v$. Thus, there are $k(k-1)$ ways to colour $v$ and $u$, and all possibilities will occur equally often within the colourings counted by $P_{\mathbf{G}_2}(k)$. Hence, given a colouring of $\mathbf{G}_1$, there will be $P_{\mathbf{G}_2}(k)/[k(k-1)]$ ways to extendthat colouring to one of all of $G$, giving us the result. ■

**Example 5.3.10 Two triangles joined along an edge, again.** As an example, we revisit our previous example. For $H$, two triangles joined along an edge, we can apply the theorm with $\mathbf{G}_1 \cong \mathbf{G}_2 \cong C_3$. Thus, we have:

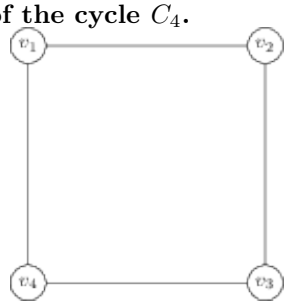
$$P_H(k) = \frac{P_{C_3}(k) P_{C_3}(k)}{k(k-1)} = \frac{k^2(k-1)^2(k-2)^2}{k} = k(k-1)(k-2)^2$$

□

### 5.3.3 Case by case analysis

The meethods of gluing and working vertex by vertex make many chromatic polynomials easy to calculate. Other graphs, however, are not amenable to our gluing theorems, and require considering several cases when working vertex by vertex.

**Example 5.3.11 The chromatic polynomial of the cycle $C_4$.** Now consider the graph $C_4$, shown at right, and suppose we try to count the number of colourings of it with $k$ colours. Vertex 1 can be any of $k$ colours, and vertex 2 has $k-1$ possibilities -- any colour except the one used for vertex 1. Moving to vertex 4, we see it is just adjacent to 1 as well, and so has $k-1$ possibilities as well.




It becomes more difficult when we try to colour vertex 3. It is adjacent to vertices 2 and 4, and so cannot be the same colour as either of these. However, vertices 2 and 4 are not adjacent, and so we don't know whether they have the same colour or not. If vertices 2 and 4, have the same colour there are $k-1$ possibilities for vertex 3, while if vertices 2 and 4 have different colours, there are only $k-2$ possibilties. Thus, we must count how many possibilities are in each of these cases.

If we want vertices 2 and 4 to have the same colour, we can first colour vertex 1 in $k$ different ways, and then pick any of the remaining $k-1$ colours for vertices 2 and 4. Then, to complete this to a colouring of $C_4$, we have to colour $v_3$, which can be any of the $k-1$ colours that aren't the colour $v_2$ and $v_4$ are coloured. Thus, the case where $v_2$ and $v_4$ have the same colour has $k(k-1)^2$ possibilities.

If we want vertices 2 and 4 to have different colours, then we can first colour $v_1$ any of $k$ colours, colour $v_2$ any of $k-1$ colours. Now, when we go to colour vertex 4 it can't be the same colour as vertex 1 since they are adjacent, and it can't be the same colour as vertex 2 by our supposition. Vertices $v_1$ and $v_2$

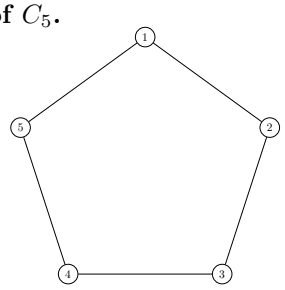
have different colours, and so this leaves $k-2$ possibilities for $v_4$. Thus there are $k(k-1)(k-2)$ possibilities to colour vertices 1, 2 and 4 so that 2 and 4 have different colours, and then there are $k-2$ possibilities left for vertex 3, giving $k(k-1)(k-2)^2$ ways to colour $C_4$ so that vertices 2 and 4 have different colours.

Adding the two cases together, this gives:

$$\begin{aligned} P_{C_4}(k) &= k(k-1)^2 + k(k-1)(k-2)^2 \\ &= k(k-1)[k-1+(k-2)^2] \\ &= k(k-1)(k^2-3k+3) \end{aligned}$$

□

**Example 5.3.12 The chromatic polynomial of $C_5$.**




A similar case-by-case argument that we made for $C_4$ works for $C_5$, except now we have a further case to deal with. Using the vertex labellings as shown to the right, note that Vertices 3 and 4 must have different colours. We will consider three cases: Vertices 1, 3 and 4 all have different colours, Vertex 1 has the same colour as Vertex 3, and Vertex 1 has the same colour as Vertex 4.

Case 1: 1, 3 and 4 have different colours. Then there are $k(k-1)(k-2)$ ways to colour vertices 1, 3, and 4, since they must all be different colours. When we go to colour vertices 2 and 5, we see that they are each adjacent to two vertices known to have different colours, and so there are $k-2$ ways to colour each of them. Thus in total, Case 1 contains $k(k-1)(k-2)^3$ colourings.

Case 2: 1 and 3 have the same colour. Then there are $k$ ways to choose this colour, and $k-1$ ways to choose a colour for vertex 2 (since 1 and 3 have the same colour), and $k-1$ ways to choose a colour for vertex 4. When we colour vertex 5, we know that 1 and 4 must have different colours, and so we have $k-2$ ways to colour vertex 5. Thus in total, Case 2 contains $k(k-1)^2(k-2)$ colourings.

By symmetry, we see that Case 3, where 1 and 4 have the same colour, is identical to Case 2. Thus, putting the three cases together, we see that:

$$\begin{aligned} P_{C_5}(k) &= k(k-1)(k-2)^3 + 2k(k-1)^2(k-2) \\ &= k(k-1)(k-2)[(k-2)^2 + 2(k-1)] \\ &= k(k-1)(k-2)(k^2-2k+2) \end{aligned}$$

□

## 5.4 Chromatic Polynomial continued

It may seem plausible that, if we consider enough cases, the case-by-case method of computing the chromatic polynomial would work for any graph, no matter how complicated. Assuming this, it would follow that the chromatic polynomial $P_{\mathbf{G}}(k)$ is in fact a polynomial. However, plausibility does not make a proof. In this section we complete the proof elegantly using induction, but first need to introduce the notion of deletion and contraction.

### 5.4.1 Deletion-Contraction

**Definition 5.4.1 Deletion.** Let $\mathbf{G}$ be a graph, and let $e \in \mathbf{G}$ be a graph. Then we use $\mathbf{G} \setminus e$ to denote the graph with the same vertex set as $\mathbf{G}$, and with all the same edges, except the edge $e$ deleted. ◊

**Definition 5.4.2 Contraction.** Let $\mathbf{G}$ be a simple graph, and let $e \in \mathbf{G}$ be an edge between vertices $v$ and $w$. Then $\mathbf{G}/e$, the graph with $e$ contracted. More precisely, $\mathbf{G}/e$ is the simple graph with vertices $V(\mathbf{G}/e) = V(\mathbf{G}) \setminus \{v, w\} \cup e$. Two non-$e$ vertices are adjacent in $\mathbf{G}/e$ if and only if they were adjacent in $\mathbf{G}$, and a vertex $y$ is adjacent to $e$ in $\mathbf{G}$ if and only if it was adjacent to $v$ or $w$ in $\mathbf{G}$. ◊

Formally, Definition 5.4.2 seems formidable, but intuitively it is not as bad as the definition looks: we are changing $\mathbf{G}$ by making the whole edge $e$ into one vertex. This may create multiple edges if a vertex was adjacent to both $v$ and $w$, and if so we simply remove any duplicate edges.

**Lemma 5.4.3 Deletion-Contraction.** *Let $\mathbf{G}$ be a simle graph, and let $e \in \mathbf{G}$ be any edge. Then we have:*

$$P_{\mathbf{G}}(k) = P_{\mathbf{G}\setminus e}(k) - P_{\mathbf{G}/e}(k)$$

*Proof.* Since $\mathbf{G}$ and $\mathbf{G} \setminus e$ have the same vertex, it is not too difficult to compare their colourings. Any valid colouring of $\mathbf{G}$ will give a colouring of $\mathbf{G} \setminus e$, but not every colouring of $\mathbf{G} \setminus e$ arises this way -- in $\mathbf{G}$, colourings require that $v$ and $w$, the two endpoints of $e$, have different colours, but there is no such requirement for $\mathbf{G} \setminus e$. So, we want to count the colourings of $\mathbf{G} \setminus e$ where $v, w$ have the same colour. But these are just the colourings of $\mathbf{G}/e$: given a colouring of $\mathbf{G}/e$m we can get a colouring of $\mathbf{G}$ by giving most vertices the same colour, and giving both $v, w$ whatever colour $e$ was. By definition, we see that $v$ and $w$ will have the same colour in this colouring; and given any colouring of $\mathbf{G} \setminus e$ where $v, w$ have the same colour we can get a colouring of $\mathbf{G}/e$ by colouring $e$ the colour that $v, w$ were. ■

**Example 5.4.4 Chromatic polynomial of $C_4$, again.** Let us compute $P_{C_4}(k)$ a different way as an illustration of how deletion-contraction works. No matter which edge of $C_4$ we pick, $C_4 \setminus e$ will be the path graph $P_4$, which is a tree, and hence we know has chromatic polynomial $k(k-1)^3$. Similarly, we have that $C_4/e = C_3$, and we know $P_{C_3}(k) = k(k-1)(k-2)$. Hence we have:

$$\begin{aligned} P_{C_4}(k) &= P_{C_4\setminus e}(k) - P_{C_4/e}(k) \\ &= P_{P_4}(k) - P_{C_3}(k) \\ &= k(k-1)^3 - k(k-1)(k-2) \\ &= k(k-1)\left((k-1)^2 - (k-2)\right) = k(k-1)(k^2 - 3k3 + 3) \end{aligned}$$

□

Since $\mathbf{G} \setminus e$ and $\mathbf{G}/e$ both have fewer edges than $\mathbf{G}$, it follows that we can repeatedly apply Deletion-Contraction to $\mathbf{G}$ until we have no edges left at all, and hence that Deletion-Contraction can compute the chromatic polynomial of any graph. In practice, this can be quite a complicated and formidable way to compute, but in the next section we show that this method can elegantly prove the chromatic polynomial is always a polynomial, and in some cases give nice formulas for this polynomial.

### 5.4.2 Chromatic polynomial is a polynomial

We first prove that the chromatic polynomial is actually a polynomial, but iterative use of deletion-contraction.

**Theorem 5.4.5** *Let* $\mathbf{G}$ *be a simple graph with* $n$ *vertices and* $m$ *edges. Then* $P_{\mathbf{G}}(k)$ *is a polynomial of degree* $n$ *in* $k$*, and moreover:*

$$P_{\mathbf{G}}(k) = k^n - mk^{n-1} + \textit{lower order terms}$$

*Proof.* The proof follows from induction on $m$, the number of edges, using deletion-contraction for the inductive step.

As a base case, we take $m = 0$. Then if $\mathbf{G}$ has $n$ vertices it must be the empty graph $E_n$. We have seen that $P_{E_n}(k) = k^n$, which indeed has the correct form needed for the theorem.

Now, for the inductive step, we assume that for any graph $H$ with $\ell < m$ edges and $n$ vertices, we have $P_H(k) = k^n - \ell k^{n-1} + \text{lower order terms}$.

Now let $\mathbf{G}$ be any graph with $m$ edges. We can assume $m > 0$, as the case where $m = 0$ is the base case; this means that $\mathbf{G}$ has at least one edge $e$. We apply deletion-contraction to the edge $e$.

If we delete $e$, the resulting graph $\mathbf{G} \setminus e$ still has $n$ vertices, but it now has $m - 1$ edges. Since this is less than $m$, we know by the inductive hypothesis that

$$P_{\mathbf{G}\setminus e}(k) = k^n - (m-1)k^{n-1} + \text{lower order terms}$$

If we contract $e$, the resulting graph $\mathbf{G}/e$ has $n-1$ vertices, and we don't know exactly how many edges it has (contracting it may create multiple edges that need to be deleted), but it has at most $m - 1$ edges, and so by the inductive hypothesis we know that

$$P_{\mathbf{G}/e}(k) = k^{n-1} + \text{lower order terms}$$

Thus, applying deletion-contraction we have:

$$\begin{aligned} P_{\mathbf{G}}(k) &= P_{\mathbf{G}\setminus e}(k) - P_{\mathbf{G}/e}(k) \\ &= k^n - (m-1)k^{n-1} + \text{lower order terms} - \big(k^{n-1} + \text{lower order terms}\big) \\ &= k^n - mk^{n-1} + \text{lower order terms} \end{aligned}$$

which is what we needed to show to finish the inductive step. ■

We end by showing that sometimes the inductive method of iteratively using deletion-contraction can compute explicit formulas for the chromatic polynomials of an infinite family of graphs.

**Lemma 5.4.6 Chromatic polynomial of $C_n$.** *Let* $C_n$ *be the cycle graph on* $n$ *vertices. Then:*

$$P_{C_n}(k) = (k-1)^n + (-1)^n(k-1)$$

*Proof.* We will induct on $n$. As a base case, when $n = 3$, we have:

$$\begin{aligned} (k-1)^3 + (-1)^3(k-1) &= k^3 - 3k^2 + 3k - 1 - (k-1) \\ &= k^3 - 3k^2 + 2k \\ &= k(k-1)(k-2) \\ &= P_{C_3}(k) \end{aligned}$$

For the inductive step, we assume that the proposition holds for $n-1$ and want to prove that it holds for $n$. We will compute $P_{C_n}(k)$ by deletion-contraction.

If we delete any edge of $C_n$, we get the path graph $P_n$, and we know

If we contract any edge of $C_n$, we get $C_{n-1}$, and we know by the inductive hypothesis that

Plugging these into deletion-contraction gives:

$$\begin{aligned} P_{C_n}(k) &= P_{C_n \setminus e}(k) - P_{C_n / e}(k) \\ &= P_{P_n}(k) - P_{C_{n-1}}(k) \\ &= k(k-1)^{n-1} - \big((k-1)^{n-1} + (-1)^{n-1}(k-1)\big) \\ &= (k-1)^{n-1}(k-1) - (-1)^{n-1}(k-1) \\ &= (k-1)^n + (-1)^n(k-1) \end{aligned}$$

as was desired. ■

## 5.5 Exercises

**1.** Eleven football games are to be arranged among eight teams A to H as follows.

| | | |
|---|---|---|
| $A$ plays $F, G, H$ | $D$ plays $C, E, G$ | $F$ plays $H$ |
| $B$ plays $E, F, H$ | $E$ plays $G$ | |

If no team can play more than once a week, what is the minimum number of weeks needed to schedule all the games? Justify your answer.

**2.** Eight students A--H each have to choose two courses from a list of eight options 1--8. They choose as follows.

| | | | |
|---|---|---|---|
| $A : 1, 2$ | $B : 2, 6$ | $C : 3, 5$ | $D : 4, 6$ |
| $E : 5, 7$ | $F : 7, 8$ | $G : 5, 8$ | $H : 3, 8$ |

You have to timetable the examinations in such a way that no student has to take two exams on the same day. What is the smallest number of days you need, and in how many ways can you fit the exams into these days? Describe one way.

**3.** Eight foods A to H are to be put in refrigerated compartments in a supermarket. Because of various regulations, some cannot share the same compartment with others, as indicated by crosses in the following table.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $A$ | × | −− | −− | × | × | −− | × |
| | $B$ | × | −− | −− | × | −− | × |
| | | $C$ | × | −− | × | × | × |
| | | | $D$ | × | × | × | −− |
| | | | | $E$ | × | × | −− |
| | | | | | $F$ | −− | −− |
| | | | | | | $G$ | × |
| | | | | | | | $H$ |

Determine the smallest number of compartments needed to display the foods and find a possible placing of the foods in the compartments.

**4.** For the graph **G** shown below, find $\chi(\mathbf{G})$ and $\chi'(\mathbf{G})$

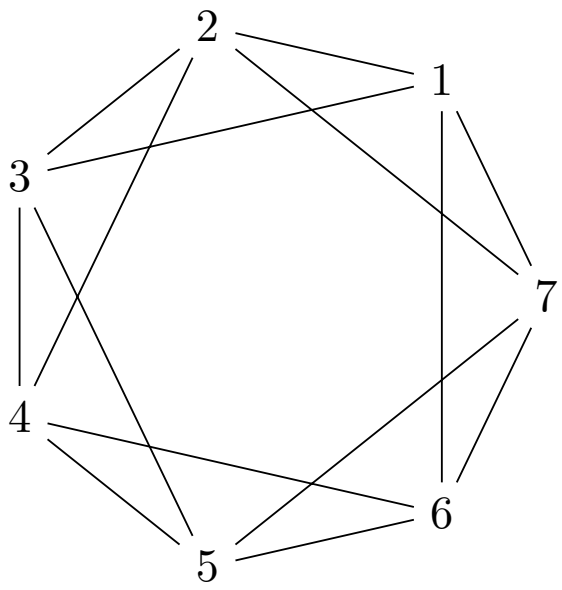
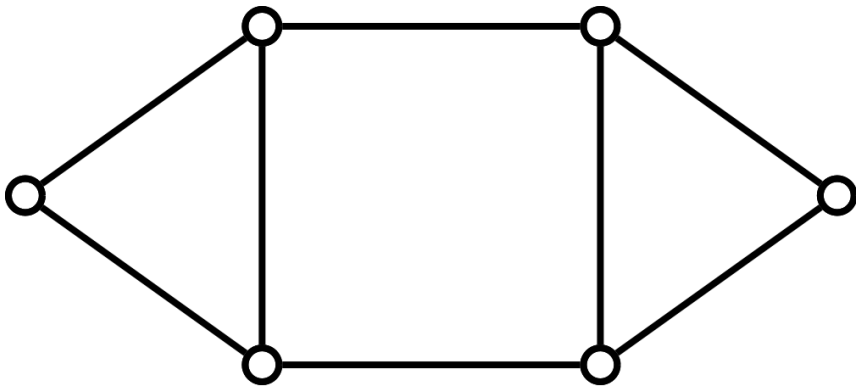
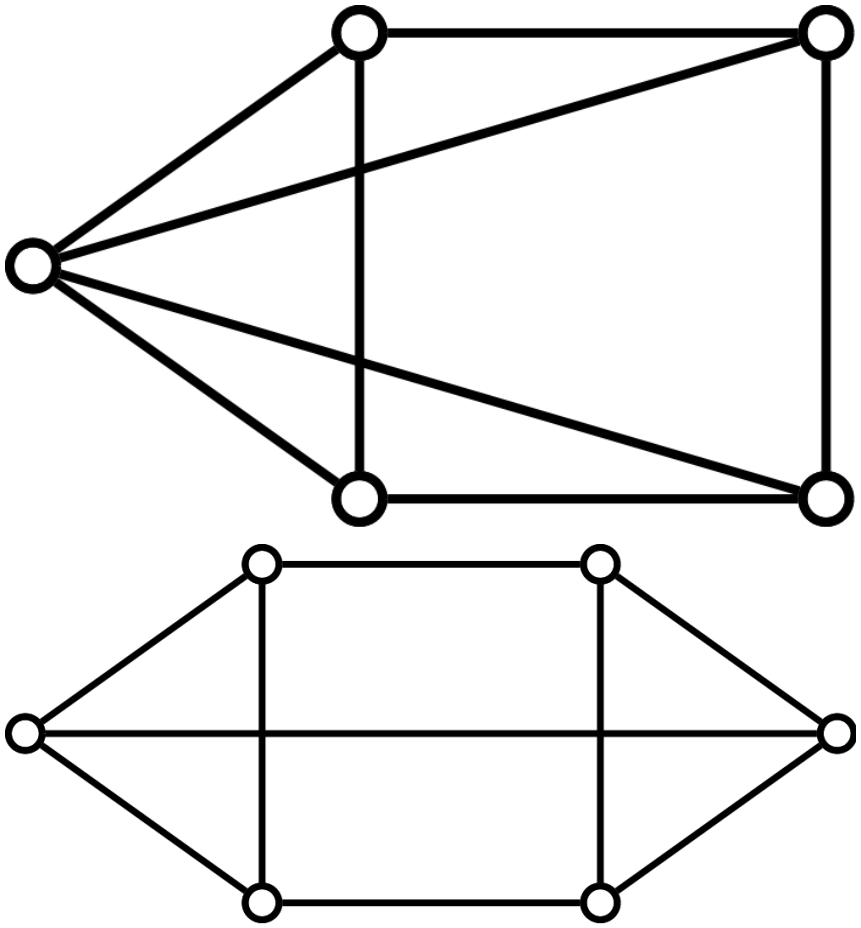



**Figure 5.5.1** The graph **G**

**5.** Find the chromatic polynomial of the following three graphs. You should use the chromatic polynomial of the four cycle as a given: $\chi_{C_4}(k) = k(k-1)(k^2-3k+3)$





**6.** Let $e$ be any edge of $K_n$. Derive the chromatic polynomial of $K_n \setminus e$ by colouring 'vertex by vertex'. Also find the chromatic polynomial of $K_n/e$, and then check that the deletion-contraction formula holds in this csae.

Now let $f \neq e$ be another edge of $K_n$. What's the chromatic polynomial of $K_n \setminus \{e, f\}$? Does it matter whether $e$ and $f$ share a vertex?