Graph Theory lecture notes

1 Definitions and examples

1–1 Definitions

Definition 1.1. A **graph** is a set of points, called **vertices**, together with a collection of lines, called **edges**, connecting some of the points. The set of vertices must not be empty.

If G is a graph we may write $V(G)$ and $E(G)$ for the set of vertices and the set of edges respectively. The number of vertices (sometimes called the "order" of G) is written $|G|$, and the number of edges (sometimes called the "size" of G) is written $e(G)$.

A graph may have multiple edges, i.e. more than one edge between some pair of vertices, or loops, i.e. edges from a vertex to itself. A graph without multiple edges or loops is called **simple**. Many natural problems only make sense in the setting of simple graphs.

If two vertices are joined by an edge they are called **adjacent**.

Definition 1.2. For each vertex v, the set of vertices which are adjacent to v is called the **neighbourhood** of v, and written $\Gamma(v)$. A **neighbour** of v is any element of the neighbourhood.

Remark. Normally we do not include v in $\Gamma(v)$; it is only included if there is a loop.

Definition 1.3. The **degree** of a vertex v, written $d(v)$, is the number of ends of edges which connect to that vertex.

In other words if we wrote down as a list the corresponding pair of vertices for every edge, the degree of a vertex is the number of times it appears in that list. As both ends of a loop go to the same vertex, each loop contributes 2 to the degree. In a simple graph, $d(v) = |\Gamma(v)|$.

What happens if we add up the degrees of all the vertices?

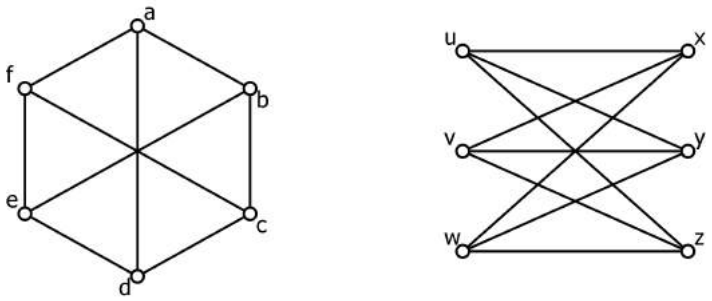
Lemma 1.4 (Euler's handshaking lemma)**.** The sum of the degrees of the vertices of a graph is equal to twice the number of edges.

We may write the sum of the degrees in two equivalent forms. Let $d_1, d_2, \ldots$ be the degrees of vertices in G and n_i be the number of vertices with degree i. Then $d_1 + d_2 + d_3 + \cdots = n_1 + 2n_2 + 3n_3 + \cdots$; the handshaking lemma tells us that each is equal to twice the number of edges. In particular, both sums are even.

In order for two graphs to be the same, they must have the same set of vertices and the same set of edges. This is very restrictive, and normally all we are interested in is whether they are *essentially* the same, that is to say they are different labellings of the same basic structure. What follows is a formal definition of this idea.

Definition 1.5. If G and H are two graphs then an **isomorphism** between G and H is a bijection $\phi : V(G) \to V(H)$ such that for any $v, w \in V(G)$ the number of edges between v and w in G is the same as the number of edges between $\phi(v)$ and $\phi(w)$ in H. If such a bijection exists we say that G and H are **isomorphic** and write $G \cong H$.

Example 1.6. Give a bijection which shows that the graphs below are isomorphic.



Definition 1.7. Let G and H be graphs. We say that H is a **subgraph** of G (or G "contains" H) if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $V(H) = V(G)$ and $E(H) \subseteq E(G)$ then H is a **spanning subgraph**.

G is always a subgraph of itself. A **proper** subgraph of G is any subgraph other than G itself.

One type of subgraph of particular importance is an **induced subgraph**: H is an induced subgraph of G if $V(H) \subseteq V(G)$ and $E(H)$ is the set of edges in $E(G)$ with both vertices in $V(H)$.

1–2 Examples and types of graphs

1. Empty graphs. E_n has n vertices and no edges.

2. Complete graphs. K_n has n vertices and each vertex is connected to each other vertex by precisely one edge.

3. Paths and cycles. The path P_n is the graph with n vertices $v_1, v_2, \ldots v_n$ and $n-1$ edges $v_1v_2, v_2v_3, \ldots v_{n-1}v_n$. The cycle C_n is graph with n edges obtained from P_n by adding an edge between the two ends; it is the graph of a polygon with n sides. The wheel graph W_{n+1} is the graph obtained from C_n by adding another vertex and edges from it to all the others.

4. Platonic graphs. There are five platonic graphs corresponding to the five platonic solids. See the additional sheet for full details. We can define graphs similarly for other polyhedra; there are 13 graphs which correspond to archimedean solids, for example.

5. Disjoint union of graphs. From any two graphs G_1 and G_2 we can form the disjoint union $G_1 \sqcup G_2$ which consists of separate copies of G_1 and G_2, with no edges between them.

   Definition 1.8. A graph is **connected** if it cannot be written as a disjoint union of two graphs.

6. Regular graphs. A regular graph is one in which all the vertices have the same degree.

   Definition 1.9. G is ***k*-regular** if every vertex has degree k.

   "Cubic" is another word for 3-regular.

7. Bipartite graphs.

   Definition 1.10. A graph is **bipartite** if we can colour the vertices red and blue in such a way that each edge connects a red vertex to a blue vertex.

The **complete bipartite graph** $K_{m,n}$ has m red vertices and n blue vertices, and from every red vertex there is exactly one edge to every blue vertex.

8. The complement. Let G be a simple graph. The **complement** of G, written $\overline{G}$ or $G^{\complement}$, is the simple graph with the same vertex set as G such that two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G. The complement of the complement is the original graph, i.e. $(G^{\complement})^{\complement} = G$.

   Examples of complements relating graphs we have already seen are $\overline{K_n} = E_n$ and $\overline{K_{s,t}} = K_s \sqcup K_t$.

2 Connectivity and Trees

2–1 Connectivity

Recall that a graph is disconnected if it is the disjoint union of two other graphs, and connected otherwise. We will find it convenient to think of connectivity in terms of being able to get from any vertex to any other.

Definition 2.1. A *walk* in a graph is a sequence of the form $v_1, e_1, v_2, e_2 \dots v_r$, for some $r \geqslant 1$, where the v_i are vertices and e_i is an edge from v_i to v_{i+1} for each $1 \leqslant i < r$.

Remark. If $r = 1$ the walk consists of a single vertex and no edges; this is still valid.

Definition 2.2. A *trail* is a walk in which no edge appears more than once.

Definition 2.3. A *path* is a walk in which no vertex appears more than once.

We might want to know whether there is a path (or trail, or walk) between specific vertices x and y. First we note that the answer is the same in each case.

Lemma 2.4. The following are equivalent:

1. there is a path from x to y;
2. there is a trail from x to y;
3. there is a walk from x to y.

Proof. First note that, since any path is also a trail and any trail is also a walk, 1) $\Rightarrow$ 2) $\Rightarrow$ 3). So it suffices to prove that 3) $\Rightarrow$ 1). We will complete the proof in lectures. $\square$

Write $x \dashrightarrow y$ if there is a path from x to y; recall that the single vertex x is a path of length 0 so $x \dashrightarrow x$. The relation $\dashrightarrow$ is an equivalence relation, since if $x \dashrightarrow y$ and $y \dashrightarrow z$ then we can put the two paths together to get a walk from x to z, and by Lemma 2.4 this implies $x \dashrightarrow z$. Consequently we may partition the vertices into equivalence classes of $\dashrightarrow$, and we call these the components of G.

Theorem 2.5. A graph G is connected if and only if there is a path between every pair of vertices.

Proof (non-examinable). Write V for the set of vertices. Suppose there is no path from v to w, for $v, w \in V$. Let C be the component containing v. Certainly $v \in C$ and $w \in V \setminus C$, so both sets are non-empty. There are no edges between C and $V \setminus C$, since there is no path from any vertex in C to any vertex not in C, by definition. So, writing G_1 for the subgraph induced by C and G_2 for the subgraph induced by $V \setminus C$, $G = G_1 \sqcup G_2$.

Conversely, suppose $G = G_3 \sqcup G_4$ is not connected. Let v be a vertex of G_3 and w be a vertex of G_4. If there is a path in G from v to w then let x be the last vertex on the path which belongs to G_3, and y be the next vertex (which exists since $x \neq w$). Then xy is not an edge of G_3, since y is not in G_3, nor is it an edge of G_4, since x is not in G_4. But xy is an edge of G, so $G \neq G_3 \sqcup G_4$. □

We may therefore equivalently define a connected graph to be one which has a path between every pair of vertices; we will generally find this definition easier to work with.

2–2 Trees

We say a graph has a cycle if it has a subgraph isomorphic to C_n for some n.

Definition 2.6. A **tree** is a connected graph with no cycles. A **forest** is a graph with no cycles.

Remark. A tree must be a simple graph.

Drawing some trees suggests that they all have the same number of edges. We'll prove this in two stages.

Theorem 2.7. A connected graph with n vertices has at least $n - 1$ edges.

Proof. We use induction on the number of vertices. When $n = 1$ there is nothing to prove. We assume the theorem is true for $n = k - 1$ and show that this implies it is also true for $n = k$. We will complete the proof in lectures. □

Theorem 2.8. Let G be a connected graph with n vertices. Then G is a tree if and only if $e(G) = n - 1$.

Proof. We have two things to prove: "if" and "only if". For the first, it is sufficient to show that if G is not a tree, so G has a cycle, then it has more than $n - 1$ edges. For the second, we show by induction on n that any tree with n vertices has $n - 1$ edges. We will complete this in lectures. □

If G is a tree, we refer to a vertex of G with degree 1 as a **leaf**.

Example 2.9. Show that any tree with at least two vertices has at least two leaves.

Let G be a connected graph on n vertices. A **spanning tree** for G is a subgraph which is a tree containing all vertices of G. Such a subgraph must exist.

We can define a simple algorithm to identify a spanning tree:

1. if there are only $n - 1$ edges remaining then we have found a tree, so stop;
2. the graph is connected and not a tree, so find a cycle;
3. delete any edge from this cycle;
4. go to step 1.

The spanning tree found is not unique because of the choice we have in step 3. If G is a graph and T a specified spanning tree then we call the edges of T **branches** and the remaining edges of G **chords**. Each chord lies in a cycle whose other edges are branches; this cycle is unique. These cycles (one for each chord) are called the **fundamental cycles** of T.

3 Applications of trees

3–1 Kruskal's algorithm

Definition 3.1. A **weighted graph** is a graph where each edge has a positive number (or "weight") associated with it.

The weights typically represent distance, time or cost of travel between vertices. The weight of a subgraph is the sum of the weights of all the edges which are included.

Kruskal's algorithm finds a minimum-weight spanning tree in a weighted graph. A typical application is that we have several towns which we wish to connect up by adding water pipes between pairs of towns, minimising the total length of pipe used; we have a table of distances between pairs of towns. Since any connected graph which is not a tree will contain a spanning tree which uses less pipe, the answer will always be a tree. So we are looking for a spanning tree with the minimum possible total edge length. This need not be unique; there may be several different spanning trees but with the same total edge length. Our table of distances might be

A					
3	B				
5	2	C			
1	4	9	D		
12	13	9	2	E	
10	7	6	9	6	F

The algorithm to find a minimal spanning tree is based on the following observation.

Lemma 3.2. Suppose the edge between A and B is at least as short as every other edge. Then there exists a minimal spanning tree which includes the edge AB.

We will prove this in lectures

Our algorithm is therefore:

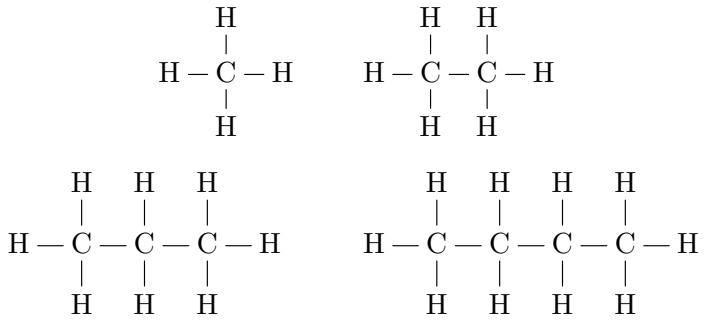
1. list the edges in increasing order of length;
2. consider the smallest remaining edge;
3. if we can add that edge to T without creating a cycle, do so, otherwise discard it;
4. if we have $n-1$ edges, T is a spanning tree so stop;
5. otherwise go back to step 2.

Example 3.3. Carry out this algorithm for the distances given above.

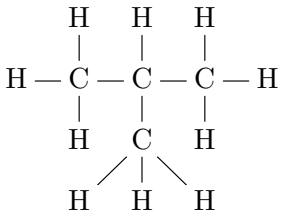
Note that this assumes that the only permitted vertices are the towns specified. In a real-world scenario we might be able to do better. For example, suppose towns A, B and C are arranged in an equilateral triangle of side length 10 miles. The minimal spanning tree has length 20 miles, but we can do better by placing a pumping station somewhere in the middle and connecting it to each town. What is the minimum length of pipe needed if we do this?

3–2 Chemistry

Alkanes have chemical formula C_nH_{2n+2}. The first four ($n = 1, \ldots 4$) are methane, ethane, propane and butane, shown below.



However, there is another compound, isobutane, which also has the formula C_4H_{10}.



Butane and isobutane are called **structural isomers**; they have the same formula but the atoms are connected in a different way. The number of isomers increases rapidly; $C_{25}H_{52}$ has over one million isomers.

We can relate the isomers of C_nH_{2n+2} to graphs. Because of the physical properties of the atoms, each carbon atom must be bonded to 4 other atoms (chemists say carbon has "valency" 4) and each hydrogen atom to 1 other atom, and the whole molecule must be connected. So we have a connected graph with n vertices of degree 4 and $2n + 2$ vertices of degree 1, so $2e = 4n+(2n+2)$ and so $e = 3n+1$, which is one less than the number of vertices. Consequently the graph is a tree.

All the hydrogen atoms are leaves; if we remove them the remaining carbon vertices must form a smaller tree. So if we want to know how many isomers there are, we need to count the number of non-isomorphic trees on n vertices, with no vertex having degree higher than 4.

Example 3.4. How many isomers of pentane (C_5H_{12}) are there?

Exercise 3.5. How many isomers of hexane (C_6H_{14}) are there?

We might also consider compounds which have some other atoms. Chlorine (Cl) has valency 1, oxygen (O) has valency 2, and nitrogen (N) has valency 3. We can again work out if the graph is a tree, and draw the trees with the appropriate number of vertices, but a tree will typically give more than one isomer since we will need to decide which vertex is the nitrogen atom, say.

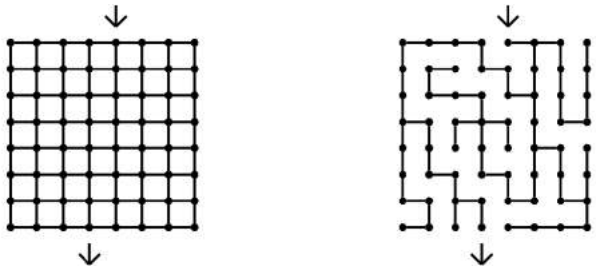
Example 3.6. Does C_3H_8O form a tree? How many isomers does it have?

3–3 Bracings

See the first homework sheet.

3–4 Mazes

A maze can be regarded as a graph where the edges are open passages and the vertices are locations, with special "start" and "end" vertices. Often they are based on a square grid, with edges removed if they are blocked by walls, hedges, etc. So we start from a graph as on the left of the diagram and remove some edges to get a subgraph such as the maze on the right.

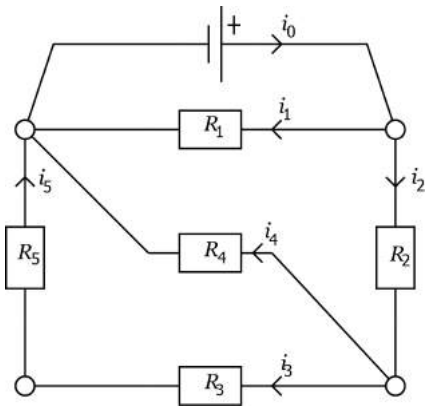


A maze is called "perfect" if every point in the maze can be reached from any other and if the path between any two points is unique. The maze above is perfect. These conditions are equivalent to the maze being a spanning tree of the original grid graph. Random maze generation algorithms typically exploit this fact; there is an algorithm to generate a perfect maze which is a randomised version of Kruskal's algorithm, for example.

Since a perfect maze is a spanning tree, if the original grid was $n \times m$ there will be nm vertices and so $nm - 1$ edges.

Example 3.7. Find and prove a relationship between the numbers of dead ends (including, possibly, the start and end vertices), T-junctions and crossroads in a perfect maze resulting from a rectangular grid.

3–5 Electrical networks



The diagram gives a simple electrical network. A voltage X is applied across the battery, and currents $i_0, \ldots, i_5$ flow in the wires (where a negative current simply means a current flowing in the other direction). The resistances $R_1, \ldots, R_5$ are known and we wish to work out the currents.

We have the following laws of electrical networks to help us.

0. Going across the battery in the positive direction produces a voltage increase of X, i.e. a voltage drop of $-X$.

1. Ohm's law. The voltage drop if a current i flows through a resistor of resistance R is iR.

2. Kirchoff's first law. The sum of currents entering a node is equal to the sum of currents leaving it. This gives us four equations. However, these equations are not linearly independent and any one of them may be deduced from the other three. So we have three equations (in six unknowns).

3. Kirchoff's second law. The sum of voltage drops around any cycle is 0. Together with Ohm's law this gives us several additional equations – one for each cycle.

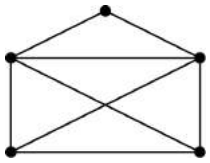
To avoid redundancy in Kirchoff's second law we should take a system of fundamental cycles; this will produce linearly independent equations. Any spanning tree will have three fundamental cycles, so gives us three more equations for a total of six linearly independent equations in six unknowns. These will have a unique solution. In general if there are n vertices and e edges we have $n-1$ equations from the first law and $e-(n-1)$ fundamental cycles, so a total of e equations.

Example 3.8. Pick a spanning tree and write down six linearly independent equations for the circuit shown. Solve them when $X = 240\,\mathrm{V}$, $R_1 = R_2 = R_3 = R_4 = 20\,\Omega$ and $R_5 = 30\,\Omega$.

4 Eulerian graphs

4–1 Euler trails

Is it possible to draw the graph shown without lifting the pen from the paper, or going over the same line twice?



Recall that a walk is a route round the edges and vertices of a graph, and a trail is a walk in which the edges used are distinct. A trail (or walk) is **closed** if it starts and ends at the same vertex. A closed trail is not necessarily a cycle, since it may visit the same vertex several times.

Definition 4.1. A graph is **Eulerian** if it has a closed trail which includes every edge. A graph is **semi-Eulerian** if has a trail which is not closed but which includes every edge.

We refer to a closed trail which includes every edge as an Euler trail. Euler observed that, provided the graph is connected, we can tell whether it is Eulerian, semi-Eulerian, or neither just by looking at the degrees of the vertices. It is easy to see that an Eulerian graph must have all its degrees even, but harder to show that a connected graph which satisfies this property is indeed Eulerian.

Lemma 4.2. If G is a graph with all vertices having even degree, then the edges of G may be partitioned into disjoint cycles.

We will prove this in lectures

Next we show that for a connected graph we can stitch together these cycles into an Euler trail.

Theorem 4.3 (Euler)**.** A connected graph G is Eulerian if and only if every vertex has even degree.

We will prove this in lectures

We get an equivalent condition for semi-Eulerian graphs by observing that a graph is semi-Eulerian if and only if we make it Eulerian by adding a single edge.

Theorem 4.4 (Euler)**.** A connected graph G is semi-Eulerian if and only if exactly two vertices have odd degree.

We will prove this in lectures

Fleury's algorithm finds an Euler trail in an Eulerian graph, or a trail which uses every edge in a semi-Eulerian graph.

Definition 4.5. A **bridge** in a connected graph is an edge whose removal will disconnect the graph. In an unconnected graph, a bridge is an edge whose removal will disconnect the component it is in.

Fleury's algorithm proceeds as follows. If the graph is Eulerian, start at any vertex and move along any edge, deleting that edge once you have crossed it, but only crossing a bridge if there is no alternative. If it is semi-Eulerian, start at either vertex of odd degree and then apply the same algorithm.

4–2 The Chinese Postman problem

A postman delivers letters to a set of streets which form the edges of a connected graph G; the vertices of G are junctions between streets. He knows the length of each street (different streets may have different lengths). He needs to start and finish at the same junction, and travel along a walk which covers every street at least once. What is the shortest walk he can take which achieves this?

The problem was originally studied by Mei-Ko Kwan (1962), who gave a necessary and sufficient condition for a shortest route. Edmonds and Johnson subsequently (1973) gave an efficient algorithm for finding the optimal closed walk.

The postman can certainly do no better than the total length L of all the streets, and a route of this length is precisely an Euler trail so he can do this if and only if G is Eulerian. If it is not Eulerian he will need to travel along some streets more than once.

Theorem 4.6. If the graph is semi-Eulerian, x and y are the two vertices of odd degree, and the shortest path between x and y has length P then the shortest postman route has length $L + P$.

Proof. The postman can certainly achieve this, by walking along a trail from x to y which covers every edge and then back to x along a shortest path.

To show that he can do no better, it is sufficient to show that in any closed walk which covers every edge, the set of edges covered twice include a path from x to y. We will complete the proof in lectures. □

We shall see an efficient algorithm to find the shortest path in a later lecture. The general case, where G may have several vertices of odd degree, is much harder.

5 Hamilton cycles

Recall that a cycle in a given graph is a subgraph isomorphic to C_n for some n, and a path in a graph is a subgraph isomorphic to P_m for some m (or, equivalently, a walk in which no vertex appears more than once).

Definition 5.1. A **Hamilton cycle** in a graph is a cycle which includes every vertex, and a **Hamilton path** is a path which includes every vertex.

Definition 5.2. A graph is **Hamiltonian** if it has a Hamilton cycle, and **semi-Hamiltonian** if it is not Hamiltonian but does have a Hamilton path.

In paths and cycles all the vertices have to be distinct (unlike trails where only edges have to be distinct). So a Hamilton path or cycle visits every vertex exactly once; an Eulerian trail uses every edge exactly once. It is much harder to determine whether a given graph is Hamiltonian than whether it is Eulerian.

Some examples of Hamiltonian graphs are the complete graph K_n (for $n \geqslant 3$) and the complete bipartite graph $K_{n,n}$ (for $n \geqslant 2$), all five platonic graphs, and (harder) the "knight's graph" whose vertices are the squares of a chessboard and whose edges connect pairs of vertices which are a knight's move apart.

Exercise 5.3. Verify that the dodecahedron and icosahedron are Hamiltonian.

Some examples of graphs which are not Hamiltonian are the complete bipartite graph $K_{n,m}$ for $n \neq m$ (since any cycle in a bipartite graph must alternate between red and blue vertices, so must have the same number of each) and any graph with a bridge (since once we cross the bridge we can't get back).

There is no simple if-and-only-if condition for a graph to be Hamiltonian. There are some sufficient conditions, most of which say that a graph with plenty of edges everywhere is Hamiltonian. The first of these was proved by G. A. Dirac (1952).

Theorem 5.4 (Dirac)**.** If G is a simple graph with $n \geqslant 3$ vertices and minimum degree at least $\frac{n}{2}$ then G is Hamiltonian.

We will not give Dirac's original proof, but instead deduce Dirac's theorem from the subsequent result proved by Ore (1960).

Theorem 5.5 (Ore)**.** Let G be a simple graph with $n \geqslant 3$ vertices. Suppose that every pair of distinct non-adjacent vertices (u, v) satisfies the inequality $d(u) + d(v) \geqslant n$. Then G is Hamiltonian.

Proof. Outline; we will fill in the details in lectures.

1. Suppose G is not Hamiltonian. We may assume G is semi-Hamiltonian (why?).
2. Take a Hamilton path $v_1 \cdots v_n$, and imagine it running from left to right. Show that there is some edge on the path whose right-hand vertex is adjacent to v_1 and whose left-hand vertex is adjacent to v_n.
3. Show how we may form a Hamilton cycle given such an edge. □

Example 5.6. What is the smallest number of edges that will ensure that a simple graph on 10 vertices is Hamiltonian?

We can deduce Dirac's theorem immediately from Ore's theorem: if $d(v) \geqslant \frac{n}{2}$ for every v then $d(u) + d(v) \geqslant n$ for every pair of vertices, so G is Hamiltonian. The $\frac{n}{2}$ in Dirac's theorem is best possible.

Exercise 5.7. Any graph can be categorised as Eulerian (E), semi-Eulerian (SE), or neither (NE) and also as Hamiltonian (H), semi-Hamiltonian (SH), or neither (NH). Combining these gives 9 categories of graphs; draw a simple connected example for each.

6 The Travelling Salesman problem

A travelling salesman needs to visit each of a group of cities, returning to his starting point. The distance from each city to each other is known, and he wishes to minimise his total distance. This corresponds to finding a Hamilton cycle of minimum weight in a weighted complete graph. This is a hard problem. There are $(n-1)!$ Hamilton cycles to choose from, and the factorials grow quite quickly. When $n = 10$ we have $9! = 362880$; when $n = 20$ we have $19! \approx 10^{17}$; and when $n = 71$ we have $70! \approx 10^{100}$.

Suppose we have 5 towns with distances given by the following table.

A				
15	B			
16	12	C		
6	19	5	D	
20	8	14	7	E

Since we only have a few towns we could just list all the routes. But is there a quicker way to get bounds on the minimum distance? Just picking any route, say $ABCDE$, gives us an upper bound, in this case $15 + 12 + 5 + 7 + 20 = 59$. But this might not be very good; how might we go about picking a route which is likely to give a good bound?

The following is a heuristic algorithm for finding an upper bound, based on the principle of nearest insertion. At each stage we have a cycle through some set of vertices which we expect to be reasonably short, and we insert another vertex to increase the length of the cycle, doing this until we have a cycle through all the vertices.

1. List the edges in increasing order of weight.
2. Start with a single vertex, A, say, and find the shortest edge from A to another vertex.
3. Start from the cycle of length 2 which goes along that edge and back again.
4. Find the shortest edge from a vertex already in the cycle to a vertex not yet in the cycle.
5. Add that edge to the cycle by inserting the new vertex after the vertex which was already in the cycle.
6. Repeat steps 4 and 5 until the cycle goes through all the vertices.

Example 6.1. Carry this out for the distance table above, starting at A.

Finding a lower bound is not as simple as finding an upper bound, since we have to be able to prove that every Hamilton cycle is at least as long. The following algorithm gives a reasonable lower bound.

1. Pick a vertex, say A, and remove it.

2. Use Kruskal's algorithm to find a minimum spanning tree for the remainder.

3. Find the two shortest edges from A to the rest of the graph.

4. Add these two lengths to the total length of the spanning tree to get a lower bound.

Lemma 6.2. This gives a lower bound on the travelling salesman problem.

We will prove this in lectures

Example 6.3. Find a lower bound for the distance table above, removing A.

In both cases the bound we get will depend on the choice of vertex we make at the start.

7 Directed graphs and directed Hamilton paths

A **directed graph**, or **digraph**, is like a graph, but instead of a set of edges between pairs of vertices we have a set of **arcs**, where each arc goes from one vertex to another (or back to itself). We draw arcs as arrows. We might think of these as a system of one-way streets. If we wanted to represent a 2-way street we would do this by having an arc from a to b and another arc from b to a.

A directed path in a directed graph is a sequence of arcs of the form $v_1 \to v_2$, $v_2 \to v_3$, ..., $v_{k-1} \to v_k$, where the v_i are distinct vertices. In other words it corresponds to our previous notion of a path but all the arcs must be going in the right direction. A directed cycle is likewise a sequence $v_1 \to v_2$, $v_2 \to v_3$, ..., $v_{k-1} \to v_k$, $v_k \to v_1$.

Consider the following puzzle. Points a to g are arranged around a circle clockwise. From a, d or f we are allowed to move exactly 3 spaces either clockwise or anticlockwise (so from a we could go to d or e); from b or e we are allowed to move 2 spaces in either direction; and from c or g we are allowed to move one space in either direction. Can we move around following these rules and visit every point exactly once?

This is asking for a Hamilton path in a directed graph obtained by drawing arcs $a \to d$ and $a \to e$, $b \to g$ and $b \to d$, and so on. There is no efficient algorithm, but we might try to systematically search for a solution either by breadth-first or depth-first searching.

7–1 Depth-first search

Choose a vertex to start from, a, say. Next choose any unvisited vertex reachable from a, and go there. Keep going, at every stage choosing an unvisited vertex which we are allowed to move to – mark any vertex from which we have only one option. Keep going until you either find a Hamilton path or are stuck – when you are stuck backtrack to the previous vertex where there was a choice and choose differently.

Example 7.1. Find a solution to the puzzle above using depth-first search.

7–2 Breadth-first search

We aim to build up a tree of possible routes. Start with a in the first column, then put the possible vertices you can get to from a in the next column, then the possible ways to continue these paths in the third column, and so on. We will find all possibilities this way.

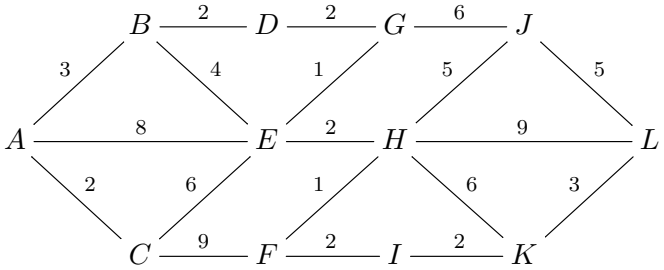
Example 7.2. Find all solutions to the puzzle above using breadth-first search.

Exercise 7.3. Arrange points a to k in a circle clockwise. From g or j you may move one step in either direction; from d or f exactly two steps in either direction; from b, e or h you may move exactly three steps in either direction; from a, c or k, four; and from i, five. Find a route through all the vertices starting at a.

8 Shortest and longest paths

8–1 Shortest paths

Recall that a weighted graph (or digraph) is one where each edge (or arc) has a positive "weight", which we typically think of as representing distance, time or cost of travel, associated with it. It is natural to ask for the shortest (in terms of total weight) path between two points in a graph or digraph; we have already seen an application of this in the Chinese Postman problem. How might we go about finding the shortest path from A to L in the following weighted graph?



Dijkstra's algorithm finds the shortest distance to every vertex. We will not give a formal proof that it works. At each step of the algorithm there will be some vertices marked with distances, and we need to distinguish which are final answers and which are merely bounds (which may change). We typically do this by circling the final answers.

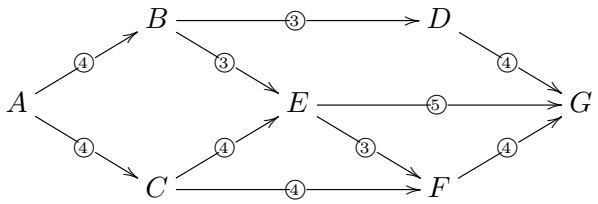
1. Start by marking the start vertex as distance 0, and circle it.
2. For each vertex adjacent to the start vertex, we calculate a bound which is the length of the edge to that vertex.
3. Find the vertex with the smallest bound among vertices which do not have final answers circled. Circle that bound, and mark the edge which gave that bound.
4. For each edge leaving the vertex you have just marked with a final answer, add that answer to the length of an edge to get a bound for the vertex that edge goes to. If it is smaller than the current bound at that vertex, replace the old bound with this one.
5. Repeat steps 3 and 4 until all vertices have their exact distances marked.

Example 8.1. Carry out Dijkstra's algorithm on the graph above.

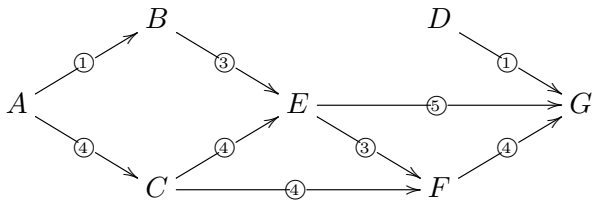
The edges we've marked form a tree and the shortest path from A to any given vertex is the path contained in that tree.

8–2 Minimum-cost maximum flows

We have a directed graph where the arcs represent pipes. Water flows along the pipes from A to G. The circled numbers on the arcs are capacities; they represent how much water can flow through the given pipe.

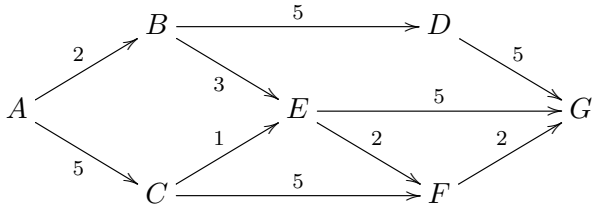


We want to send as much water as possible through the network. In order to do this we choose a path through the network, then put as much water as we can through that path. So we might start by putting 3 units along the path $ABDG$. This uses up all the capacity of BD, and AB and DG have 1 unit of capacity left. Then we repeat the process on the network with these reduced capacities, shown below (any arc which can't take any more water is removed).



Continue adding flow until there are no paths from A to G left. We next might put 4 units of flow along $ACFG$. There is now only one remaining path through the network, $ABEG$, and we can put 1 unit through it. This gives us our final flow, which is 8 units in total. This will always give the maximum possible amount of flow.

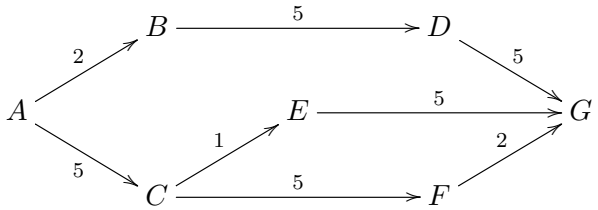
However, there are lots of different ways to put 8 units through this network. In applications some arcs may be cheaper to use than others; we give each arc a cost, which is a positive number. The cost of n units of water flowing through a pipe of cost c is nc. The costs might be as follows.



$ABDG$ has cost 12, $ACFG$ 12 and $ABEG$ 10, so the total cost of our flow is $12 \times 3 + 12 \times 4 + 10 \times 1 = 94$. We want to find a flow which still puts the maximum amount of water (8 units) through the network, but is as cheap as possible. One approach to doing this is that at each stage when we are looking for a path to add, we use Djikstra's algorithm to find the shortest path in terms of costs.

Remark. This algorithm is not guaranteed to find the absolute minimum cost for every network; there is a slightly more complicated version which will always do so.

Example 8.2. Use the minimum-cost maximum flow algorithm on the network given above, and find the total cost of the resulting flow. The second time we run the shortest path algorithm, it will be on the network shown below.



8–3 Longest paths

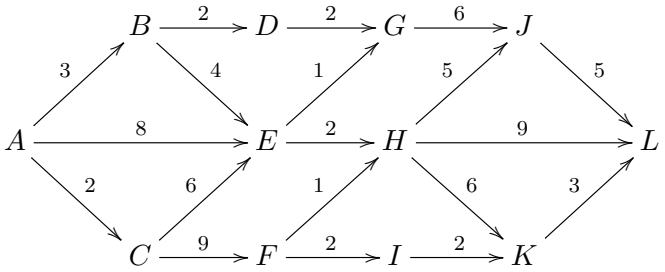
For the longest path problem we are generally interested in directed graphs with no cycles. With that restriction, the following algorithm will find a longest path.

1. Mark the desired start vertex as having distance 0.
2. Find an unmarked vertex X which only has arcs in from vertices already marked with distances.
3. For each arc coming into X, add the weight of the arc to the distance marked at the vertex the arc comes from; each of these is a possible path length.
4. Mark X with the maximum of these.
5. Mark the arc which gave the maximum value as being used.
6. If all vertices are marked, stop.
7. Otherwise, return to step 2.

The distance marked at a vertex will be the length of the longest path to that vertex. The longest path will be the path to that vertex from the start which consists entirely of marked arcs. This will not always be unique: we may have a tie for the maximum in step 3.

This algorithm will work provided we can always find a suitable vertex in step 2. It can fail for one of two reasons: either there is a vertex which cannot be reached from the start, or there is a cycle.

Example 8.3. Run the algorithm on the following directed graph (a directed version of the graph we used for the shortest path algorithm) to find the longest paths from A to each other vertex, and their lengths.



8–4 Activity networks

This is an application of the longest path algorithm. We have a list of tasks, a length of time each task will take, and a set of precedence relations which tell us which tasks must be completed before we can start which other tasks.

Our tasks might be as follows.

A	prepare foundations	7 days
B	make and position door frame	2 days
C	lay drain and floor base	15 days
D	install services and fittings	8 days
E	erect walls	10 days

F	plaster ceiling	2 days
G	erect roof	5 days
H	install and paint doors and windows	8 days
I	fit gutters and pipes	2 days
J	paint inside	3 days

The precedence relations are that D must come after E, E must come after A and B, F must come after D and G, G must come after E, H must come after G, I must come after C and F, and J must come after I.

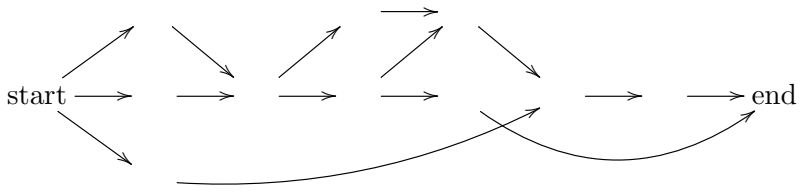
What is the shortest time in which the structure can be built? We can solve such problems using Fulkerson's algorithm.

1. Obtain a directed graph to represent the project. Put "start" in the first column. Find the tasks which do not have any prerequisites and put them in the second column. Add arcs of weight 0 from "start" to each of them. In each subsequent column put all the tasks which only have to follow tasks which already appear in earlier columns, and add an arc for each precedence. Once all the tasks are included, put "end" in a final column and add an arc to "end" from every task which nothing has to follow. Each arc should be labelled with the length of time taken by the task it is coming from. "Start" can be thought of as a task which takes no time but must occur before any other.

2. Use the longest path algorithm to find all the longest path distances from "start". The minimum time for completion is the length of the longest path to "end", and the earliest time at which each task may be begun is the length of the longest path to that task. If we start the tasks at those times then we will always have completed all prerequisites by the time we are due to start a given task.

3. If required, we may also work out the latest starting times (that is, the latest time a task can start if we are still to finish in the minimum time) by going backwards, column by column. The latest time for "end" is the same as the earliest time. To get the latest time for a task X, look at all the latest times of tasks which must follow X. Then subtract the time taken for X from the smallest of these.

Example 8.4. Carry out all three parts of the algorithm for the tasks given above. The directed graph we get will look like the following (labels omitted).



Remark. Some tasks give us no leeway: the earliest start time equals the latest start time. We refer to the longest path from "start" to "end" as the critical path. All tasks on such a path will have equal start and end times, and so any delay on the critical path will delay the entire project. The critical path might not be unique.

9 Planar graphs

An old puzzle concerns three houses which each need to be connected to three utilities. Is it possible to connect each house to each utility without crossing the connections?

Definition 9.1. A **planar graph** is a graph which may be drawn in the plane without its edges crossing. A **plane graph** is a specified drawing of a graph in the plane without edges crossing.

In graph-theoretic terms, the puzzle is asking whether $K_{3,3}$ is planar.

Theorem 9.2. The graphs K_5 and $K_{3,3}$ are not planar.

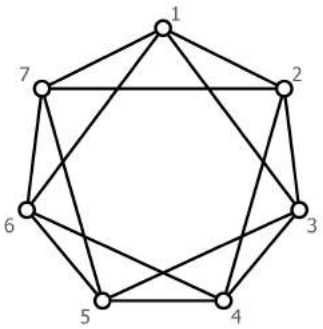
We will prove this in lectures

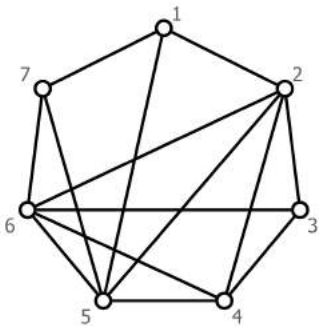
9–1 The planarity algorithm for Hamiltonian graphs

The proof we will give generalises to a way to check whether a Hamiltonian graph G is planar, provided we know or can easily find a Hamilton cycle.

1. Find a Hamilton cycle. Draw the vertices and edges of this cycle as a regular polygon, and draw the remaining edges as straight lines inside the polygon.
2. Draw the incompatibility graph, H. To do this, for each edge of G inside the cycle, give H a vertex labelled by that edge; connect two vertices of H if the corresponding edges of G cross in the drawing.
3. G is planar if and only if H is bipartite. If H is bipartite then we can draw G in the plane by putting the edges of G which correspond to red vertices of H inside the Hamilton cycle, and those which correspond to blue vertices of H outside. Conversely, given a plane drawing of G we can colour vertices of H red if they correspond to edges inside the cycle and blue otherwise, and this will be a bipartition of H.

Example 9.3. Carry out the algorithm on the two graphs shown; step 1 has already been carried out.

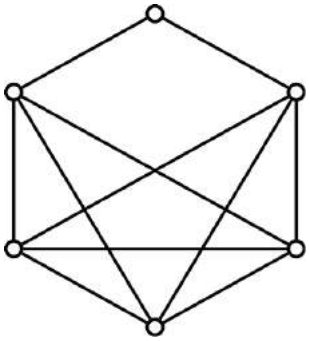




9–2 Kuratowski's theorem

It is easy to see that any subgraph of a planar graph is also planar – start from a plane drawing of the original graph and rub out edges and vertices as appropriate. So any graph which contains a non-planar graph is not planar, and in particular any graph which contains K_5 or $K_{3,3}$ is non-planar.

K_n is non-planar if $n \geqslant 5$ (but is planar if $n < 5$) and $K_{r,s}$ is non-planar if $r, s \geqslant 3$ (but is planar if $r < 3$). However, there are many non-planar graphs which do not contain either K_5 or $K_{3,3}$ as a subgraph. The graph shown does not contain K_5 (or $K_{3,3}$), but it looks like K_5, and is non-planar for the same reason. This example motivates the following definition.



Definition 9.4. A **subdivision** of a graph G is any graph obtained by inserting new vertices of degree 2 along some of the edges; each edge can have any number of such vertices added.

In other words H is a subdivision of G if we can obtain H by replacing some of the edges of G with paths. We interpret "some of the edges" above to include the case where none of the edges are modified, so that G is a subdivision of itself.

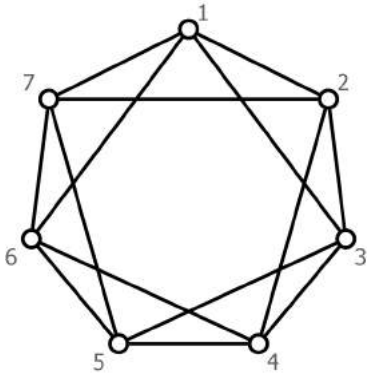
Suppose H is a subdivision of G. If G is planar then we may take a plane drawing of G and add new vertices along the edges to get a plane drawing of H. Conversely, if H is planar we may get a plane drawing of G by replacing the path of each subdivided edge by a single curve.

Since K_5 and $K_{3,3}$ are not planar, neither is any subdivision of either of them, and so neither is any graph which contains a subdivision of K_5 or $K_{3,3}$. Remarkably, that is enough to give a complete classification of planar graphs.

Theorem 9.5 (Kuratowski)**.** A graph is planar if and only if it contains no subdivision of K_5 or $K_{3,3}$ as a subgraph.

We say two graphs G, H are **homeomorphic** if both are isomorphic to subdivisions of the same graph F. If H is a subdivision of G then G and H are homeomorphic (by taking $F = G$), but in general two graphs can be homeomorphic without either being isomorphic to a subdivision of the other. Kuratowski's theorem may be equivalently stated as "A graph is planar if and only if it contains no subgraph homeomorphic to K_5 or $K_{3,3}$." We shall not prove Kuratowski's theorem here as the proof that any graph without such a subgraph is planar is very complex.

Example 9.6. We previously used the planarity algorithm to show that the graph to the right is not planar. Consequently, by Kuratowski's theorem, it must contain a subdivision of K_5 or $K_{3,3}$. Find such a subdivision.



9–3 Euler's formula

Any plane graph G divides the plane into regions, called the **faces** of G. The region outside G is counted as one of the faces. The **degree** of a face is the number of sides of edges bounding it (so that if an edge has the same face on both sides, it is counted twice in the degree). Like vertex degree, this satisfies a handshaking lemma.

Lemma 9.7. The sum of the face degrees of a plane graph is twice the number of edges.

Euler's formula gives a relation between the number of edges, the number of faces and the number of vertices.

Theorem 9.8 (Euler)**.** If a plane connected graph has v vertices, e edges and f faces then $v + f - e = 2$.

We will prove this in lectures

For a fixed planar connected graph G, with v vertices and e edges, it follows that every plane drawing of G has $e - v + 2$ faces, so the number of faces of G does not depend on how it is drawn in the plane.

Since the graph of any polyhedron is planar, the same formula applies to a polyhedron with v vertices, e edges and f faces. (Imagine a hollow rubber polyhedron. Puncture any face, stretch the hole and flatten the polyhedron onto the plane. The punctured face becomes the external face of a plane drawing of the graph.)

We can use Euler's formula to bound the number of edges a planar graph can have.

Corollary 9.9. If G is a simple connected planar graph with $v \geqslant 3$ vertices and e edges then $e \leqslant 3v - 6$. If additionally G has no cycle of length 3 then $e \leqslant 2v - 4$.

We will prove this in lectures

Both bounds are best possible.

Example 9.10. Use these bounds to give another proof of the fact that K_5 and $K_{3,3}$ are not planar.

9–4 Fullerenes

A fullerene is a polyhedral molecule consisting entirely of carbon atoms. The first such molecule to be synthesised, buckminsterfullerene, has 60 carbon atoms arranged in the form of a truncated icosahedron (the polyhedron used as a basis for the standard football design).

In any fullerene each atom is bonded to three others, and every face is either a hexagon or a pentagon. We can think of this as a 3-regular plane graph with p faces of degree 5 and h faces of degree 6 (and no others).

Example 9.11. Prove that every fullerene has exactly 12 pentagonal faces.

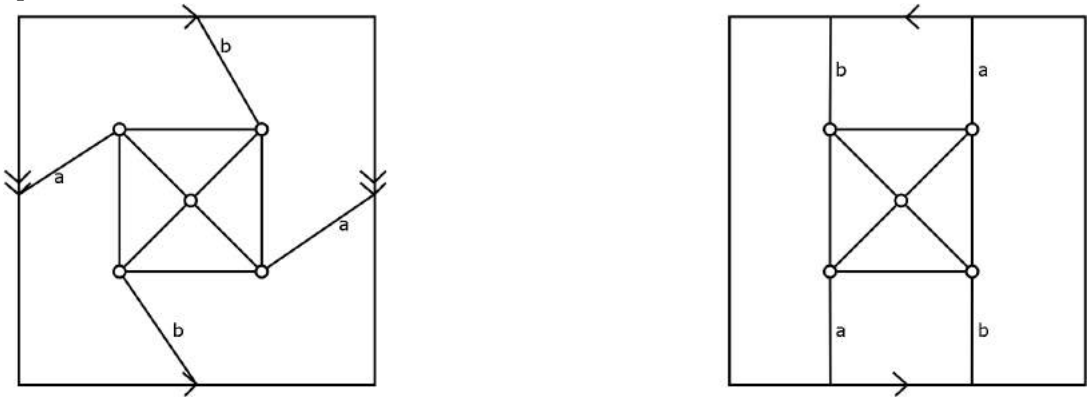
A 20-carbon fullerene with 12 pentagonal faces and no hexagonal faces does exist, as do many other fullerenes smaller than buckminsterfullerene, but they do not occur naturally whereas it does (and so do four others which are slightly larger). One reason for this is that configurations where two pentagonal faces share an edge are much less stable. What is the smallest fullerene in which no two pentagons share an edge? In fact no two pentagons can share a vertex, since the three faces meeting at each vertex all share edges, so the 12 pentagons must between them span 60 different vertices. Buckminsterfullerene has 60 carbons and no two pentagons share an edge, so it is the smallest such fullerene.

9–5 Graphs on surfaces

What happens if instead of trying to draw graphs in the plane without edges crossing we try to draw them on some other surface? The graphs which can be drawn on a sphere without edges crossing are just the planar graphs. Other surfaces, such as the torus, Möbius strip or Klein bottle, might be more interesting.

We can represent the torus as a square with opposite sides identified – so that moving off the left-hand side of the square brings us back onto the right-hand side, and likewise with the top and bottom. We may represent the Möbius strip as a square with top and bottom identified with a twist.

The diagram below shows how K_5 may be drawn on the torus and Möbius strip without edges crossing. Edges which cross the sides of the squares have been marked to show how they join up.



Example 9.12. Draw $K_{3,3}$ on the torus and Möbius strip without edges crossing. For each of the four drawings, count the number of faces. Is Euler's formula satisfied?

Definition 9.13. The **orientable surface of genus** g is the torus-like surface with g holes.

The orientable surface of genus 0 is a sphere, that of genus 1 is a normal torus, and so on.

Definition 9.14. The **genus** of a graph G is the smallest value of g such that G can be drawn on the orientable surface of genus g without edges crossing. We write $g(G)$ for the genus of G.

Since any graph which can be drawn on the surface of genus g can also be drawn on the surface of genus $g+1$, we may equivalently say that G has genus g if it may be drawn on the orientable surface of genus g without edges crossing, but not on the orientable surface of genus $g-1$. Euler's formula works for the plane or sphere, i.e. for genus 0, but may be generalised to higher genus. We won't prove this generalisation.

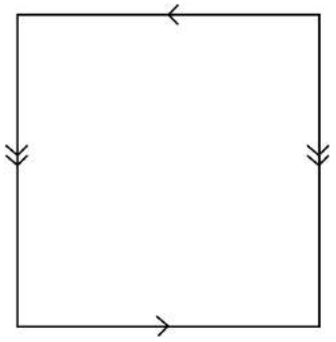
Theorem 9.15. If G is a connected graph of genus g with v vertices and e edges, and is drawn on the orientable surface of genus g without crossings in such a way as to have f faces, then $v + f - e = 2 - 2g$.

The assumption that G has genus g is necessary; if we remove that condition then we instead get $v + f - e \geqslant 2 - 2g$.

Example 9.16. Let G be a simple connected graph of genus g with e edges and $v \geqslant 3$ vertices. Show that $g \geqslant 1 - v/2 + e/6$. Use this inequality to show that $g(K_n) \geqslant (n-3)(n-4)/12$ (for $n \geqslant 3$).

In fact $g(K_n) = \lceil \frac{1}{12}(n-3)(n-4) \rceil$ for every $n \geqslant 3$ (where the notation $\lceil x \rceil$, pronounced "ceiling of x", means the smallest integer y with $y \geqslant x$). So $g(K_3) = g(K_4) = 0$, $g(K_5) = g(K_6) = g(K_7) = 1$, $g(K_8) = 2$.

One other surface we might consider is the Klein bottle, which is like a torus with a twist; we can represent it as shown.



Example 9.17. Draw K_6 on the Klein bottle.

10 Vertex colouring

Let G be any graph. A **vertex colouring** of G is an assignment of a colour to each vertex such that every edge goes between two different colours. We often just call this a colouring of G.

We will generally focus on simple graphs when we talk about vertex colouring. If G has a loop then it has no colourings at all, and it makes no difference for colouring purposes whether we have one or two (or more) edges between x and y.

We say that G is k-colourable if it has a colouring which uses at most k colours. Note that a graph is bipartite if and only if it is 2-colourable; the definitions are the same.

Theorem 10.1. A graph is bipartite if and only if it has no cycle of odd length.

We will prove this in lectures; we need the following lemma

Lemma 10.2. Any closed walk with an odd number of edges contains an odd cycle.

Proof. Let H be the graph on the same vertex set with an edge between x and y for every time x and y appear consecutively in the walk (so if we use the same edge xy of G twice, there are two edges between x and y in H). All vertices have even degree in H, so by Lemma 4.2 the edges of H may be partitioned into disjoint cycles. If all these cycles are even then H would have an even number of edges, which it doesn't, so at least one must be odd. □

Definition 10.3. The **chromatic number** of G, written $\chi(G)$, is the smallest k such that G is k-colourable.

If $\chi(G) = k$ we sometimes say "G is k-chromatic".

10–1 Bounds on the chromatic number

Let G be a simple graph. Choose an ordering of the vertices of G. We will colour the vertices one at a time in order with colours from the set $\{1, 2, 3, \ldots\}$. When colouring each vertex, we choose the smallest colour which has not already been assigned to one of its neighbours. This is the **greedy algorithm** for colouring. It will always produce a colouring, but the number of colours used will depend on what order we colour the vertices in. There is always some order for which the greedy algorithm uses only $\chi(G)$ colours. On the other hand, the worst-case performance of the greedy algorithm can be very bad: for each n there is a bipartite graph on $2n$ vertices and an ordering for which greedy uses $n + 1$ colours.

Theorem 10.4. If G is a simple graph of maximum degree Δ then $\chi(G)$ is at most $\Delta + 1$. If $\chi(G) = \Delta + 1$ then G must have at least $\Delta + 1$ vertices of degree Δ.

We will prove this in lectures

The following theorem of Brooks, which we will not prove, is a stronger version of this.

Theorem 10.5 (Brooks)**.** If G is a simple connected graph with maximum degree Δ then either $G = K_{\Delta+1}$, G is an odd cycle, or $\chi(G) \leqslant \Delta$.

10–2 Colouring on surfaces

A natural question is to ask whether we can bound the chromatic number of planar graphs. It was conjectured as early as 1852 that only four colours are needed to colour any simple plane graph, and several fallacious proofs were produced during the 19th century. In 1890, Heawood showed that an earlier "proof" by Kempe was incorrect, but adapted the method to prove that five colours are always sufficient. The conjecture itself was not proved until 1976, by Appel and Haken – the famous "four-colour theorem". Their proof relied on complicated arguments to reduce the conjecture to almost 2000 different cases, which were then checked by computer.

It is quite easy to show that six colours are sufficient, using a simple consequence of Corollary 9.9.

Lemma 10.6. Any simple planar graph has a vertex of degree at most 5.

Corollary 10.7. If G is a simple plane graph then $\chi(G) \leqslant 6$.

We will prove these in lectures

This method forms the basis of the next two results (both due to Heawood).

Theorem 10.8. If G is a simple plane graph then $\chi(G) \leqslant 5$.

Proof. Again, we use induction on the number of vertices; the result is true for graphs with 5 or fewer vertices. Suppose G has n vertices and assume the result is true for all graphs with fewer vertices. By Euler's formula, G has some vertex v with $d(v) \leqslant 5$. If we remove v the remaining vertices may be coloured with 5 colours; take any such colouring. If this colouring does not use all 5 colours on the neighbours of v then there is a colour available for v and we are done. If not then v has 5 neighbours all of different colours. Label these $a_1, \ldots a_5$ going clockwise around v in our plane drawing, and assume a_i has colour i for each i.

To complete the proof, we need to show that we can modify the colouring so that there will be a colour available to use at v. We will do this part in lectures. □

For planar graphs the lower bound (that 4 colours may be required) is trivial, but the upper bound (the four-colour theorem) is very hard. For other orientable surfaces, however, the upper bound is relatively easy whereas showing that this many colours may be required is hard. We will just prove the upper bound. Here $\lfloor x \rfloor$, "floor of x", is the largest integer less than or equal to x.

Theorem 10.9 (Heawood's bound)**.** If G is a simple graph drawn on the surface of genus g, where $g > 0$, then $\chi(G) \leqslant \left\lfloor \frac{1}{2} \left(7 + \sqrt{1 + 48g}\right) \right\rfloor$.

Proof (non-examinable). Write $h(g) = \left\lfloor \frac{1}{2}\left(7 + \sqrt{1+48g}\right)\right\rfloor$. Again we prove the result by induction. As before, it is sufficient to show that any simple graph on this surface has a vertex of degree at most $h(g) - 1$. Suppose not; take a counterexample G. From Euler's formula for the surface, we can deduce $e \leqslant 3v + 6g - 6$ (the equivalent of Corollary 9.5 for this surface). Since every vertex has degree at least $h(g)$, by handshaking $2e \geqslant h(g)v$, so

$$v(h(g) - 6) \leqslant 12g - 12\,.$$

Since $g \geqslant 1$, $h(g) - 6 > 0$. Also, since G is simple and any vertex has degree at least $h(g)$, we must have $v \geqslant h(g) + 1$. So

$$(h(g) + 1)(h(g) - 6) \leqslant 12g - 12\,.$$

Now $h(g) > \frac{1}{2}\left(7 + \sqrt{1+48g}\right) - 1$, and consequently

$$\left(\sqrt{1+48g} + 7\right)\left(\sqrt{1+48g} - 7\right) < 48g - 48\,,$$

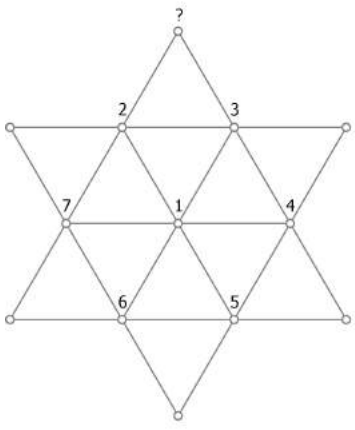
but in fact the two sides are equal. □

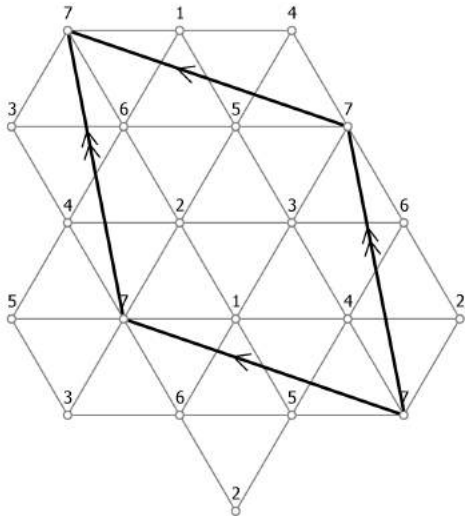
Heawood conjectured that this bound was best possible for every g, i.e. that for every $g \geqslant 1$ there exists a graph of genus g which requires $h(g)$ colours. This is now known – it is the Ringel–Youngs Theorem – and the last remaining cases were proved in 1968. Heawood also proved a similar bound for graphs which may be drawn on non-orientable surfaces, and conjectured that it was best possible in every case. This conjecture is false for the Klein bottle (in 1930 Franklin showed that any graph drawn on the Klein bottle can be coloured with 6 colours, whereas Heawood's bound was 7), but has since been proved for every other non-orientable surface. The analogue of Euler's formula on the Klein bottle is that $v + f - e \geqslant 0$ (with equality if the graph cannot be drawn on a simpler surface).

Theorem 10.10 (Franklin)**.** If the simple graph G can be drawn on the Klein bottle then $\chi(G) \leq 6$.

Proof (non-examinable). We will show in lectures that if any 7-chromatic simple graph may be drawn on the Klein bottle then K_7 may be drawn on the Klein bottle in such a way that every face is a triangle.

To complete the proof we must show that if K_7 can be drawn on a surface in such a way that every face has degree 3 and is simply connected then this surface must be the torus. Pick a vertex, 1, and label its neighbours 2, ... 7 clockwise.





What face meets face 123 at the edge 23? It cannot be 237, since we already have an edge 27; it cannot be 234 either so it is 235 or 236. Without loss of generality (the other choice is equivalent to reflecting the diagram and relabelling) it is 235. Now everything else is determined, and our triangulation looks locally like the second diagram; identifying equal points makes a torus, as shown. □

11 The chromatic polynomial

Previously we have considered the question of *whether* a given graph can be coloured with k colours. Now we will think about *how many ways* there are to colour a given graph with k colours. Let G be a simple graph, and write $P_G(k)$ for the number of ways of vertex-colouring G with k colours.

For the empty graph, $P_{E_n}(k) = k^n$, since there are k choices at each vertex. For the complete graph, $P_{K_n}(k) = k(k-1)(k-2)\cdots(k-n+1)$, because the first vertex can be coloured with any colour, the second with any other colour, the third with any colour not yet used, and so on.

Lemma 11.1. If T is a tree on n vertices then $P_T(k) = k(k-1)^{n-1}$.

We will prove this in lectures

11–1 Techniques for calculating $P_G(k)$

1. Disconnected graphs. If $G = A \sqcup B$ then $P_G(k) = P_A(k)P_B(k)$, since we may colour each part separately.

2. Gluing together at a vertex. Suppose G is made up of two parts, G_1 and G_2, which share a single vertex v. This time we can't just colour G_1 and G_2 separately because we need to make sure both colourings give v the same colour. There are $P_{G_1}(k)$ k-colourings of G_1; for each one there are $\frac{1}{k}P_{G_2}(k)$ colourings of G_2 which give v the same colour, so $P_G(k) = \frac{1}{k}P_{G_1}(k)P_{G_2}(k)$.

   A special case of this rule is where G has a vertex of degree 1, and removing this vertex leaves the graph G_1. In this case after colouring G_1 we have $k-1$ choices for the other vertex, so $P_G(k) = (k-1)P_{G_1}(k)$.

3. Gluing together along an edge. Suppose G is made up of two parts, G_1 and G_2, with exactly two vertices in common, u and v, and G_1, G_2 and G all include the edge uv. Now our colourings must agree on u and v. Since u and v are adjacent, they must have different colours in any colouring of G_1 or G_2, and any pair of different colours is equally likely. Consequently $P_G(k) = \frac{1}{k(k-1)}P_{G_1}(k)P_{G_2}(k)$.

   Be careful to make sure that uv is an edge of the graph and that you include this edge in both G_1 and G_2. A special case is where G has a vertex v of degree 2 and its neigbours have an edge between them. Then removing v leaves a graph G_1 and $P_G(k) = (k-2)P_{G_1}(k)$.

4. Suppose G has subgraphs G_1 and G_2 which between them include every edge and vertex of G, and $G_1 \cap G_2 \cong K_r$. Then $P_G(k) = P_{G_1}(k)P_{G_2}(k)/P_{K_r}(k)$. This is a generalisation of the previous rules, but is less often useful for higher r. The intersection being a complete graph is necessary for any rule of this type to work.

11–2 The contraction–deletion relation

What is $P_{C_4}(k)$? None of the above rules help, so we will need a new idea. To that end we will define several modifications to graphs.

Let G be a simple graph, and x, y be vertices. If x and y are not adjacent then $G + xy$ is the simple graph obtained by adding the edge xy; similarly, if x and y are adjacent then $G - xy$ is the simple graph obtained by deleting the edge xy. If x and y are not adjacent then we may **identify** x and y to form the simple graph $G_{x=y}$. This only has a single vertex corresponding to x and y; this is adjacent to z if and only if z was adjacent to x or y in G (but we never have multiple edges, even if z was adjacent to both). Finally, if e is any edge of G then we may **contract** e to form the simple graph G/e. To contract e, first delete e and then identify its two vertices.

Theorem 11.2. If G is a simple graph and x, y non-adjacent vertices then $P_G(k) = P_{G+xy}(k) + P_{G_{x=y}}(k)$.

Corollary 11.3. If G is a simple graph and e is any edge of G then $P_G(k) = P_{G-e}(k) - P_{G/e}(k)$.

We will prove Theorem 11.2 in lectures

Corollary 11.3 follows immediately by applying Theorem 11.2 to $G - e$.

This gives two recursive algorithms for finding the chromatic polynomial of any simple graph G. Using Theorem 11.2, we can choose two non-adjacent vertices x and y, and form the graphs $G_1 = G + xy$ and $G_2 = G_{x=y}$, so that $P_G = P_{G_1} + P_{G_2}$. If G_1 is now complete we know P_{G_1}; if not we choose two non-adjacent vertices in G_1 and form two more graphs G_3, G_4 from it in the same way. Likewise if G_2 is not complete we form graphs G_5 and G_6 from it, and now $P_G = P_{G_3} + P_{G_4} + P_{G_5} + P_{G_6}$. We continue this process until all remaining graphs are complete.

In fact, when performing this algorithm by hand we do not need to carry on until everything is complete – if a graph with an easy to calculate chromatic polynomial comes up we just leave it.

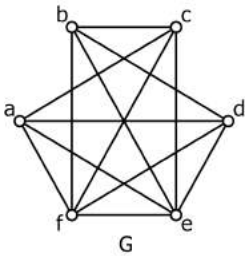
The other recursive algorithm is to use Corollary 11.3. Choose an edge e_0 of G and write $G_1 = G - e_0$, $G_2 = G/e_0$ so that $P_G = P_{G_1} - P_{G_2}$ If G_1 is not empty then we can choose another edge e_1 in G_1 and set $G_3 = G_1 - e_1$, $G_4 = G_1/e_1$, and similarly for G_2. Then we have

$$\begin{aligned} P_G &= (P_{G_3} - P_{G_4}) - (P_{G_5} - P_{G_6}) \\ &= P_{G_3} - P_{G_4} - P_{G_5} + P_{G_6}\,; \end{aligned}$$

we have to be careful to keep track of signs with this method. Continue until we have reduced to graphs whose chromatic polynomials we know – if implementing this on computer it may be easier to continue until all remaining graphs are empty.

Example 11.4. Use each method in turn to find $P_{C_4}(k)$, and check that the answers agree.

Example 11.5. Calculate the chromatic polynomials of the wheel W_5 and the graph G shown.



G

11–3 Properties of the chromatic polynomial

We are now able to use Corollary 11.3 to prove that $P_G(k)$ has the following properties.

1. $P_G(k)$ is a polynomial in k.
2. Its degree is the number of vertices of G, n.
3. The coefficients are integers.
4. The coefficient of k^n is 1.
5. The coefficient of k^{n-1} is $-e(G)$.
6. The coefficients alternate in sign.

We only need a single method to prove all these facts. We will just prove properties 1 to 5 in lectures, as including property 6 would make the proof unnecessarily fiddly (though not really any harder).

Theorem 11.6. Let G be a simple graph with n vertices and m edges. Then $P_G(k)$ is a polynomial with integer coefficients of the form $k^n - mk^{n-1} + \cdots$.

We will prove this in lectures

---

12 Edge colouring

Let G be a graph with no loops (but multiple edges are permitted). An **edge colouring** of G is as assignment of a colour to each edge such that any two edges which have a common vertex are assigned different colours. We say G is k-edge-colourable if there is an edge colouring of G using at most k colours and define the **chromatic index** of G, which we write $\chi'(G)$, to be the smallest k for which G is k-edge-colourable.

Clearly if G has maximum degree Δ then $\chi'(G) \geqslant \Delta$, since the Δ edges meeting at a vertex of maximum degree all require different colours. A greedy-algorithm based argument would quickly give $\chi'(G) \leqslant 2\Delta$, since each edge shares a vertex with at most $2\Delta - 1$ others, but if G is simple then much more is true.

Theorem 12.1 (Vizing)**.** If G is a simple graph with maximum degree Δ then $\chi'(G) \leqslant \Delta + 1$.

We will not prove Vizing's Theorem. The requirement that G is simple is necessary: for graphs with multiple edges, $\chi'(G)$ can be as large as $3\Delta/2$. If G is simple then $\chi'(G)$ is either Δ or $\Delta + 1$, but it is in general difficult to determine which. We will determine which it is in a few special cases, however.

Lemma 12.2. Let G be a Δ-regular simple graph with n vertices. If n is odd then $\chi'(G) > \Delta$.

We will prove this in lectures

A natural application of edge colouring is tournament scheduling. Imagine a league of n players in which every player plays against every other once. How many rounds are needed to schedule all the games? The answer is $\chi'(K_n)$ – the edges represent matches and the colour of an edge is the round in which that match occurs.

Theorem 12.3. If n is odd then $\chi'(K_n) = n$; if n is even then $\chi'(K_n) = n - 1$.

Proof. By Lemma 12.2, $\chi'(K_n) \geqslant n$ if n is odd, and by Vizing's Theorem $\chi'(K_n) \leqslant n$, so certainly $\chi'(K_n) = n$ if n is odd. So we just need to construct an edge colouring of K_n with $n-1$ colours when n is even to complete the proof. In fact we will give a construction with n colours for n odd as well. We will complete this in lectures. □

We can also determine the chromatic index for any bipartite graph. The proof is non-examinable, but included for interest.

Theorem 12.4. If G is bipartite with maximum degree Δ then $\chi'(G) = \Delta$.

Proof (non-examinable). This proof uses a modification of Kempe chains. We prove it by induction on the number of edges; it is certainly true for a graph with 1 edge. Suppose it is true for any bipartite graph with fewer edges than G, and remove an edge xy from G to get a bipartite graph G' with fewer edges and maximum degree at most Δ. Choose an edge colouring of G' with Δ colours and try to extend it to an edge colouring of G. Now x had degree at most Δ in G so has degree at most $\Delta - 1$ in G', so there is some colour which has not yet been used at x, 1, say. Likewise there is some colour which hasn't been used at y. If this is also 1 we are done, so suppose it is 2. We want to colour the edge xy with colour 1, so we try to swap some colours in such a way that 1 becomes available at y, while still being available at x.

Construct a path from y whose edges are alternately coloured 1 and 2 which is as long as possible. This path is unique: if we have reached z and are looking for an edge of colour 1 then there is at most one since we have a valid edge colouring. If it does not reach x then we can just swap all the colours along the path to get a new colouring in which 1 is available at x and y, which we can extend to a colouring of G. But the path cannot reach x after an odd number of edges, since then the edge reaching x would be colour 1, which we assumed was available at x. It cannot reach x after an even number of vertices either, since then these edges together with xy would be an odd cycle in G, and G is bipartite. So the path does not reach x and we can swap the colours on it as required. □

13 Face colouring

We stated the four-colour theorem as a result on vertex colouring, but it was originally conjectured as a result on face-colouring. A **face-colouring** of a bridgeless plane graph is an assignment of colours to faces such that two faces which meet along an edge have different colours. We require graphs to be bridgeless so that no face meets itself along an edge.

Theorem 13.1. A bridgeless plane graph G is face-colourable with two colours if and only if every vertex has even degree.

We will prove this in lectures

Given a connected plane graph G we may define its **dual** G^* as follows. Pick a point inside each face of G; these will be the vertices of G^*. For each edge e of G we draw a corresponding edge of G^* between the vertices corresponding to faces on either side of e, which crosses e. Note that G^* is defined only for a particular plane drawing of G, and we define not just the graph G^* but also a plane drawing of it. The drawing we use for G does matter: it is easy to construct plane graphs G and H such that $G \cong H$ but $G^* \ncong H^*$.

Provided G is connected G^* satisfies the following conditions.

a G^* is connected.

b $(G^*)^* = G$.

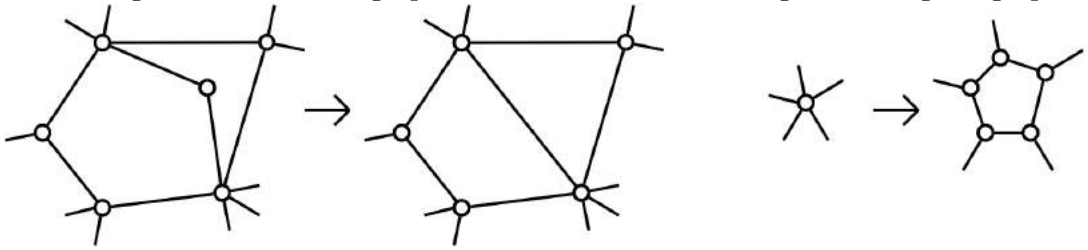
c G^* has no loops $\iff$ G has no bridges.

d If G has v vertices, e edges and f faces then G^* has f vertices, e edges and v faces.

Consequently we may state the four-colour theorem in either of the following equivalent forms.

1. Every loopless planar graph is 4-colourable.
2. Every bridgeless plane graph is 4-face-colourable.

In fact, given any bridgeless planar graph G we may construct a 3-regular bridgeless planar graph which is at least as hard to face-colour. G has no vertices of degree 1, because it is bridgeless, and any vertex of degree 2 may be replaced by a single edge connecting its neighbours, without affecting which faces meet. Any vertex of degree at least 4 may be truncated as shown; any k-face-colouring of the truncated graph contains a k-face-colouring of the original graph.



Consequently the four-colour theorem is equivalent to

3. Every 3-regular bridgeless plane graph is 4-face-colourable.

Our final result links the 4-colour theorem to edge colouring.

Theorem 13.2 (Tait)**.** The following are equivalent:

3. Every 3-regular bridgeless plane graph is 4-face-colourable.
4. Every 3-regular bridgeless plane graph is 3-edge-colourable.

The proof of this result is non-examinable; we will only prove in lectures that (3) $\Rightarrow$ (4).

The four-colour theorem means that any bridgeless planar cubic graph must be 3-edge-colourable. The planarity condition is necessary; perhaps the best-known counterexample in graph theory is the Petersen graph, which is bridgeless and 3-regular but not 3-edge-colourable (in addition to many other remarkable properties).