

# 1 Graph theory

There are many types of graphs. Graphs can be *directed* or *undirected*, they may be labeled in different ways, sometimes edges that are self-loops are allowed, sometimes there may be multiple edges between two vertices etc. The variety of definitions is caused by the situation that graphs are used in very many different applications. Previously we have already used directed graphs to represent relations. Here we begin by considering undirected graphs that are called simply *graphs*.

## 1.1 Basic definitions on (undirected) graphs

A graph  $G$  is an ordered pair  $(V, E)$  where

1.  $V$  is a non-empty set of vertices, and,
2.  $E$  is a set of unordered pairs  $\{u, v\}$  that are called the edges of  $G$  ( $u, v \in V, u \neq v$ ).

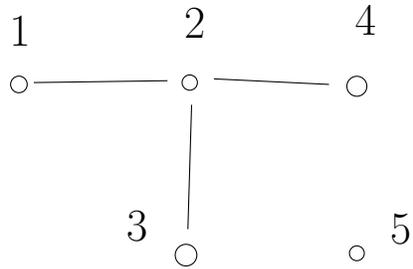
Note that in the above definition the edges are undirected, there can be at most one edge between given vertices and there are no self-loops, i.e., edges from a vertex to itself. This type of graphs are sometimes also called *simple graphs*. When referring to a “graph” we mean a simple graph.

Note that an unordered edge (i.e., edge that doesn't have a direction) is represented by a two element set  $\{u, v\}$ .

The graph is *finite* if  $V$  is finite. Although the definition does not exclude infinite graphs, unless otherwise mentioned, by a graph we always mean a finite graph.

An edge  $\{u, v\}$  can be denoted also as  $uv$  (or  $vu$ ) for short. The vertices  $u$  and  $v$  are the *end points* of the edge. We say also that the edge  $uv$  *connects* the vertices  $u$  and  $v$  or that  $uv$  *touches*  $u$  and  $v$  (or  $u$  and  $v$  are incident to the edge  $uv$ ). If  $u$  and  $v$  are connected by an edge we say that  $u$  and  $v$  are *adjacent* vertices.

Graphs can be represented in the well-known way using a drawing. In the drawing the vertices are represented by points (or small circles) on the plane and adjacent vertices are

Figure 1: A graph  $G$ .

connected by a line. Often such a figure is also called a “graph”. However, it is important to remember that a graph is completely determined by the set of vertices and the set of edges, and two very different looking figures may represent the same graph.

For computing applications various representations of graphs can be used, as discussed below in subsection 1.4. Below we begin by defining the adjacency matrix representation.

Suppose  $V = \{v_1, \dots, v_n\}$ . The *adjacency matrix* of the graph  $G = (V, E)$  is a Boolean  $n \times n$  matrix  $\mathbf{A}_G = (a_{i,j})_{n \times n}$  defined by setting for  $1 \leq i, j \leq n$ ,

$$a_{i,j} = \begin{cases} 1, & \text{if } v_i v_j \in E, \\ 0, & \text{if } v_i v_j \notin E. \end{cases}$$

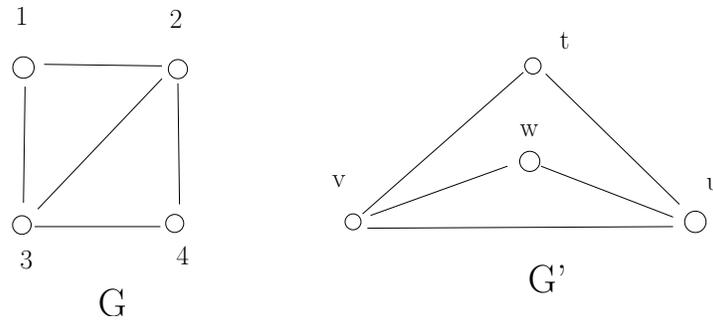
When talking about undirected graphs, the matrix  $\mathbf{A}_G$  is symmetric and all elements on the main diagonal are zeros.

**Example 1.1** *The graph  $(\{1, 2, 3, 4, 5\}, \{12, 23, 24\})$  is represented by Figure 1.*

*The adjacency matrix of the graph is*

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

While a graph  $G$  can be represented by very different looking figures, any two graphs that are represented by the same figure are the same as abstract graphs: they may differ only in the names of the vertices. This type of “similarity” of graphs is called an *isomorphism*.

Figure 2: Graphs  $G$  and  $G'$ .

**Definition 1.1** An isomorphism between graphs  $G = (V, E)$  and  $G' = (V', E')$  is a bijective function  $\varphi : V \rightarrow V'$  such that

$$\{u, v\} \in E \text{ iff } \{\varphi(u), \varphi(v)\} \in E' \text{ for all } u, v \in V.$$

The graphs  $G$  and  $G'$  are isomorphic if there is an isomorphism from  $G$  to  $G'$  and this is denoted by  $G \simeq G'$ .

It is easy to verify that graph isomorphism is

1. reflexive ( $G \simeq G$ ),
2. symmetric ( $G \simeq G'$  implies  $G' \simeq G$ ), and,
3. transitive ( $G \simeq G'$ ,  $G' \simeq G''$  implies  $G \simeq G''$ ).

Thus, the isomorphism relation partitions graphs into equivalence classes and graphs in the same equivalence class can be identified (i.e., considered to be the same graph).

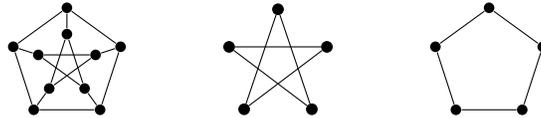
**Example 1.2** The graphs  $G$  and  $G'$  depicted in Figure 2 are isomorphic. An isomorphism  $\varphi$  between  $G$  and  $G'$  is given by  $\varphi(1) = t$ ,  $\varphi(2) = u$ ,  $\varphi(3) = v$  and  $\varphi(4) = w$ .

A graph with  $n$  vertices that has all possible  $\binom{n}{2}$  edges is the *complete graph* of size  $n$ ,  $K_n$ . The other extreme is the *null graph*  $0_n$  that has  $n$  vertices and no edges. A graph with a single vertex  $K_1$  ( $= 0_1$ ) is called the *trivial graph*.

**Definition 1.2** A graph  $G' = (V', E')$  is a subgraph of graph  $G = (V, E)$  if  $V' \subseteq V$  and  $E' \subseteq E$ . This is denoted  $G' \preceq G$ . If  $V' = V$  and  $E' \subseteq E$ , we say that the subgraph  $G'$  spans the graph  $G$ .

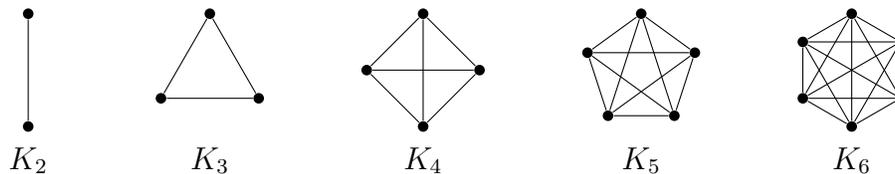
The complement of the graph  $G$  is the graph  $G^c = (V, E^c)$  where  $E^c = \{uv \mid u, v \in V, u \neq v, uv \notin E\}$ .

**Example 1.3** The leftmost graph, known as the Petersen graph, is used frequently as an example or counterexample when proving results in graph theory. Observe that the Petersen graph contains the two rightmost graphs as subgraphs.



**Definition 1.3** The degree of a vertex  $v$ ,  $d(v)$ , is the number of edges touching  $v$ . A graph is regular if all vertices have the same degree. If this degree is  $k$ , the graph is  $k$ -regular.

**Example 1.4** The complete graph  $K_n$  is  $(n - 1)$ -regular and the null graph  $0_n$  is 0-regular.

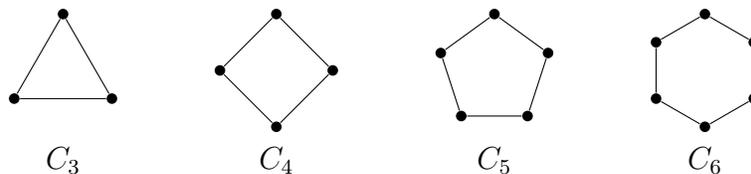


**Example 1.5** For  $n \geq 3$ , the  $n$ -cycle is the graph

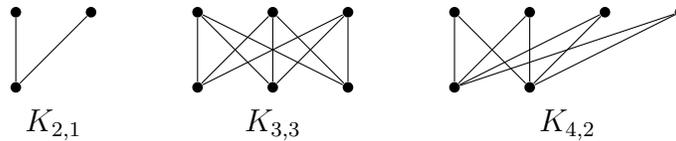
$$C_n = (\{0, 1, 2, \dots, n - 1\}, E),$$

where  $E = \{01, 12, 23, \dots, (n - 2)(n - 1), n0\}$ . The  $n$ -cycle is 2-regular.

Note that defining a cycle graph with  $n = 2$  will result in more than one edge between two vertices (a multi-graph).



A bipartite graph is a graph whose vertex set  $V$  can be partitioned into two subsets  $V_1$  and  $V_2$  such that each edge joins a vertex in  $V_1$  to a vertex in  $V_2$ . If  $V_1$  contains  $m$  vertices and  $V_2$  contains  $n$  vertices, and if each vertex in  $V_1$  is joined to every vertex in  $V_2$ , then we say that the graph is a complete bipartite graph, denoted  $K_{m,n}$ .



**Theorem 1.1** For any finite graph  $G = (V, E)$ :

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

**Proof (outline two possibilities).** The theorem can be proven using induction on the number of edges of the graph and using the fact that each edge touches two vertices and it increases both of their degrees by one. Alternatively the claim follows by observing the adjacency matrix  $\mathbf{A}_G$ : the degree of a vertex is the number of 1's on the row corresponding to the vertex, and each edge is represented in the matrix by exactly two 1's.

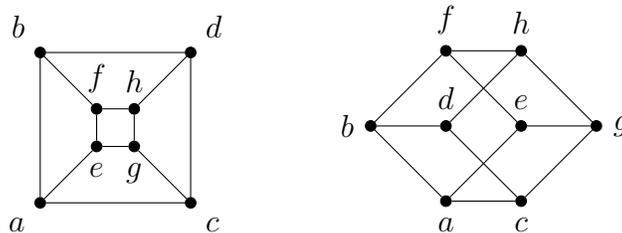
A vertex is called *even* or *odd* depending on whether the number  $d(v)$  is even or odd. From Theorem 1.1 we get:

**Corollary 1.1** A finite graph has always an even number of odd vertices.

The degrees of vertices are often useful for testing whether two graphs are isomorphic and for finding possible isomorphisms between them. If a mapping  $\varphi : V \rightarrow V'$  is an isomorphism between graphs  $G = (V, E)$  and  $G' = (V', E')$ , it must be the case that  $d(\varphi(v)) = d(v)$  for all  $v \in V$ .

### 1.1.1 More on graph isomorphism

**Example 1.6** The following two graphs are isomorphic, as evidenced by the labels of vertices in both graphs.



How do we determine when two graphs are isomorphic? We could do so by inspection, but this approach is suboptimal when dealing with graphs that contain more than a handful of vertices and edges. Taking a more formalized approach, we shift our focus away from the graphs  $G_1$  and  $G_2$  and toward the isomorphism  $f$ . If we can show that  $f$  is *edge-preserving* (which is just a more concise way of saying that  $f$  preserves the property of vertex adjacency), then we can conclude that  $f$  is an isomorphism.

We can determine whether  $f$  is edge-preserving by representing both  $G_1$  and  $G_2$  as adjacency matrices. Then,  $G_1$  and  $G_2$  are isomorphic if and only if there exists a permutation matrix  $\mathbf{P}$  such that  $\mathbf{A}_{G_1} = \mathbf{P} \cdot \mathbf{A}_{G_2} \cdot \mathbf{P}^T$ . In other words (or if you don't remember your linear algebra lectures),  $G_1$  and  $G_2$  are isomorphic if and only if we can rearrange the rows and columns of  $\mathbf{A}_{G_2}$  to get  $\mathbf{A}_{G_1}$ .

**Remark.** Note that we choose to use adjacency matrices here because the property of edge-preservation affects pairs of vertices, and the entries of an adjacency matrix relate pairs of vertices. Using an adjacency matrix over an adjacency list also gives us the benefit of being able to use matrix algebra to solve a problem.

A related question is how can we determine when two graphs are *not* isomorphic? We could take the same approach we just saw, where we essentially try to prove that we cannot permute the rows and columns of one adjacency matrix to obtain another, but this can be tedious and difficult to do by hand.

An easier approach for showing that two graphs are not isomorphic is to compare the *invariants* of each graph. An invariant of a graph is a property that identical graphs must share; if this property does not match between the graphs, then we can conclude that the graphs are not isomorphic. Three common invariants we can use to rule out isomorphism are:

- the number of vertices (since the isomorphism  $f$  must be a bijection, both graphs must contain the same number of vertices);
- the number of edges (since the isomorphism  $f$  must be edge-preserving, both graphs must contain the same number of edges); and
- the degrees of each vertex (since the isomorphism must be an edge-preserving bijection,

there must be a correspondence between vertices of the same degree in each graph).

However, we must be careful: although we can rule out isomorphism if an invariant does not hold, we cannot conclude that two graphs are isomorphic if invariants do hold.

**Example 1.7** Consider the following pair of graphs:



Both of these graphs contain 5 vertices and 6 edges, and within each graph there are two vertices of degree 3 and three vertices of degree 2. However, these graphs are not isomorphic! (To see why, observe that the right graph contains  $C_3$  as a subgraph, while the left graph does not. Indeed, no bipartite graph contains  $C_3$  as a subgraph.)

Finally, it is worth mentioning another famous problem in computer science called (appropriately) the *graph isomorphism* problem. This problem asks whether two given graphs are isomorphic to one another, but it has a number of applications outside of graph theory: for instance, the graph isomorphism problem can determine matching patterns in images, calculate matchings and measures on text strings, and construct layouts of electrical circuits on boards. Although the graph isomorphism problem has been solved for certain types of graphs, not much is known about the general problem. In fact, the general problem is one of the few problems suspected to lie *between* the classes of “easy” and “hard” problems.

## 1.2 Walks and connected components

**Definition 1.4** A walk in a graph  $G = (V, E)$  is a non-empty sequence of vertices

$$W = (v_0, v_1, v_2, \dots, v_k),$$

where  $k \geq 0$  and  $v_i v_{i+1} \in E$  for  $i = 0, 1, \dots, k-1$ . The walk  $W$  connects the vertices  $v_0$  and  $v_k$  and they are the end points of the walk. If  $v_0 = v_k$ ,  $W$  is a closed walk (and otherwise it is open). The length of  $W$  is  $k$ .

If  $v_0v_1, v_1v_2, \dots, v_{k-1}v_k$  are all distinct edges the walk  $W$  is a trail. If  $v_0, v_1, \dots, v_k$  are all distinct vertices the walk  $W$  is a path.

A closed walk  $(v_0, v_1, \dots, v_{k-1}, v_0)$  is a  $k$ -cycle if  $v_0, v_1, \dots, v_{k-1}$  are all distinct vertices. A cycle is called even or odd depending on whether  $k$  is even or odd.

In the graph of Figure 1,  $(1, 2, 3, 2, 4)$  is an open walk of length 4. It is neither a trail or a path.

The graph  $G'$  of Figure 2 has 3-cycles  $(v, t, u, v)$  and  $(v, w, u, v)$  and a 4-cycle  $(v, t, u, w, v)$ .

**Definition 1.5** Vertices  $u$  and  $v$  of a graph  $G = (V, E)$  are connected (or reachable from one another) if there is a walk whose end points are  $u$  and  $v$ . This is denoted  $u \approx v$ . The graph  $G$  is connected if any two vertices of  $G$  are connected.

Any vertex  $u$  of a graph  $G = (V, E)$  is connected with itself since  $u$  by itself is a walk of length 0. This means that the relation  $\approx$  is reflexive. It is easy to verify that  $\approx$  is also symmetric and transitive, and thus an equivalence relation.

Each equivalence class  $[v]_{\approx}$  ( $v \in V$ ) defines a connected subgraph of  $G$ ,  $([v]_{\approx}, E_v)$  where

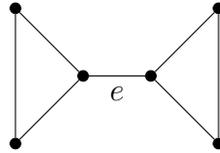
$$E_v = \{uw \in E \mid u \approx v \text{ and } w \approx v\}.$$

These subgraphs are called the *components* of  $G$ . Obviously the graph  $G$  is connected if and only if  $\approx V \times V$ , that is,  $G$  has only one component which is  $G$  itself.

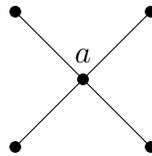
If we take a graph  $G$  with  $m$  connected components, and if removing a vertex from  $G$  produces a new graph  $G'$  with more than  $m$  connected components, then we call that vertex a **cut vertex**. If the same outcome occurs after removing an edge, then we call that edge a **cut edge**.

The problem of determining cut vertices and cut edges in a connected graph is important, since it tells us quite a bit about the structure of the graph. For instance, determining the cut edges in a graph that models a highway system will reveal weak points in the system; if the road corresponding to a cut edge becomes impassable, then traffic won't be able to get from one point to another point.

**Example 1.8** The following graph is connected, since there exists a path between every pair of vertices. If we delete edge  $e$  from the graph, then the graph will consist of two connected components, but the graph itself will no longer be connected.



**Example 1.9** The following graph is connected. If we delete vertex  $a$  from the graph, then each edge will disappear and the graph will consist of four connected components: one for each remaining vertex.



The shortest walk between two connected vertices  $u$  and  $v$  is called the *geodetic line* between  $u$  and  $v$ . Obviously a geodetic line is always a path. The length of a geodetic line between  $u$  and  $v$  is the *distance* between  $u$  and  $v$ , and it is denoted  $d(u, v)$ . If  $u$  and  $v$  are not connected, we define  $d(u, v) = \infty$ . A distance function  $d : V \times V \rightarrow \mathbb{N}$  has the usual properties required of a distance:

1.  $d(u, v) = 0$  iff  $u = v$ ,
2.  $d(u, v) = d(v, u)$ ,
3.  $d(u, v) \leq d(u, w) + d(w, v)$  (triangle inequality).

The following property is proved easily using induction on  $n$ .

**Lemma 1.1** If  $u$  and  $v$  are connected vertices of a graph with  $n$  vertices then  $d(u, v) \leq n - 1$ .

The following result is often useful. The result can be proved using induction on  $m$ .

**Theorem 1.2** Consider a graph  $G = (\{v_1, \dots, v_n\}, E)$  and let  $\mathbf{A}_G = (a_{ij})_{n \times n}$  be the adjacency matrix of  $G$ . For  $m = 1, 2, \dots$  let

$$\mathbf{A}_G^m = (a_{ij}[m])_{n \times n}$$

be the  $m$ th power of  $\mathbf{A}_G$ , where matrix multiplication is done using integers (and not as Boolean matrices).

Then  $a_{ij}[m]$  is the number of walks of length  $m$  from vertex  $v_i$  to vertex  $v_j$ ,  $1 \leq i, j \leq n$ .

Using the matrices  $(a_{ij}[m])_{n \times n}$  we can also determine whether two vertices of a graph are connected and what is their distance:

- vertices  $v_i$  and  $v_j$  are connected if and only if  $i = j$  or  $a_{ij}[m] \neq 0$  for some value  $m = 1, 2, \dots, n - 1$ , and,

- 

$$d(v_i, v_j) = \begin{cases} 0, & \text{if } i = j; \\ \min\{m \mid a_{ij}[m] \neq 0, 1 \leq m \leq n\}, & \text{if } i \neq j \text{ and } a_{ij}[m] \neq 0 \text{ for some } m, 1 \leq m \leq n; \\ \infty, & \text{otherwise.} \end{cases}$$

### 1.2.1 Applications of Connectedness

A number of famous problems in computer science relate to the notions of paths, connectedness, and cuts in graphs. A selection of the most well-known are as follows.

- The *graph reachability* problem asks whether a path exists from one given vertex to another given vertex within a graph. In an undirected graph, we can solve this problem by simply finding the connected components of the graph. In a directed graph, a number of approaches exist to solve the problem; it is possible to find the solution in time proportional to  $|V|^3$  by using the Floyd-Warshall algorithm. (Other algorithms are more efficient, but require some preprocessing work.)
- The *shortest path* and *longest path* problems, as the name suggests, ask you to find the shortest path and longest path, respectively, between two vertices in a graph. Here, the “length” of a path may be measured by the number of edges or by weights associated with each edge. An abundance of algorithms exist for solving the shortest path problem, including Dijkstra’s algorithm (which runs in time proportional to  $|V| \log(|V|) + |E|$ ) and the Bellman-Ford algorithm (which runs in time proportional

to  $|V| \cdot |E|$ ). Interestingly, however, the longest path problem has no known efficient solution.

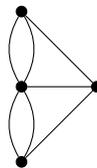
- The *min-cut* and *max-cut* problems ask you to find the minimum number of edges and maximum number of edges, respectively, that partition the vertices of a graph into two components. To find a minimum cut, we can use techniques such as the Edmonds-Karp algorithm, which runs in time proportional to  $|V| \cdot |E|^2$ . Unfortunately, we are not so lucky when it comes to finding a maximum cut: this problem once again has no known efficient solution.

### 1.3 Eulerian and Hamiltonian graphs

The below Theorem 1.3 holds also for so called multi-graphs. A *multi-graph* allows more than one edge between two vertices. Most of the above notions can be extended in an obvious way for multi-graphs. However, it should be noted that the edges of a multi-graph cannot be defined by an (unordered) pair consisting of the two end points.

**Definition 1.6** *A walk in a graph (or multi-graph)  $G = (V, E)$  is an Eulerian trail if every edge of  $G$  appears there exactly once. A closed Eulerian trail is called an Eulerian circuit. A graph is Eulerian if it has an Eulerian circuit.*

**Example 1.10** *The notion of Eulerian trail has its root in the Bridges of Königsberg problem studied by Leonhard Euler in the 18th century.*



*When Euler studied this problem, he wanted to know whether there existed a route through Königsberg that traversed each of the seven bridges exactly once. The bridge connections are abstracted in the above graph. Euler wanted to find a walk through the given graph that included each edge exactly once (that is, an Eulerian trail).*

An Eulerian circuit naturally has to visit all vertices of the graphs, however, it may visit the

same vertex multiple times. As a consequence it follows that every Eulerian graph must be connected.

On the other hand, a graph being connected is not sufficient to guarantee that it must be an Eulerian graph. The graph  $G$  of Figure 2 has an Eulerian trail  $(3, 4, 2, 3, 1, 2)$  but  $G$  is not Eulerian (i.e., it does not have an Eulerian circuit).

The next result that characterizes Eulerian graphs is perhaps the “first theorem” of graph theory. Leonhard Euler proved it originally for multi-graphs (1736).

**Theorem 1.3** *A finite connected (multi-)graph is Eulerian if and only if the degree of every vertex is even.*

**Proof.** The condition is necessary because an Eulerian circuit “arrives” and “departs” from each vertex an equal number of times and always using a different edge. The sufficiency of the condition is proved using induction on the number of edges. (Will be done in class, time permitting.)  $\square$

Now that we have a condition for Eulerian circuits, let’s try to modify this condition in order to say something about when a graph contains an Eulerian trail. The condition is modified in the sense that we may now have a certain number of odd-degree vertices in our graph.

**Theorem 1.4** *A connected graph  $G$  contains an Eulerian trail if and only if  $G$  contains exactly two vertices of odd degree.*

**Proof.** ( $\Rightarrow$ ): Assume that  $G$  contains an Eulerian trail. Every time such a trail reaches a vertex  $u$  of  $G$ , it contributes two to  $d(u)$  (one for entering  $u$  and one for leaving  $u$ ). Since we are dealing with a trail, the initial and final vertices are distinct, so both vertices have degree 1 and they are, in fact, the only vertices in  $G$  of odd degree.

( $\Leftarrow$ ): Assume exactly two vertices of  $G$  are of odd degree; say,  $u$  and  $v$ . If we join  $u$  and  $v$  by an edge, then every vertex in  $G$  will be of even degree. By Theorem 1.3,  $G$  contains an Eulerian circuit. If we remove this added edge from  $G$ , then the Eulerian circuit becomes an Eulerian trail.  $\square$

With this result, we can finally conclude why no trail through Königsberg exists that crosses each bridge exactly once. Note that, of the four vertices the graph of Example 1.10, three of the vertices have degree 3 and one vertex has degree 5. Since none of the vertices are of even degree, Theorem 1.4 tells us that no Eulerian trail can exist.

Next we consider paths and circuits that visit each vertex once.

**Definition 1.7** *A walk in a graph  $G$  is a Hamiltonian path if it visits every vertex of  $G$  exactly once. A cycle of  $G$  is a Hamiltonian cycle if it first visits every vertex of  $G$  exactly once and at the end returns to the original vertex. A graph  $G$  is Hamiltonian if it has a Hamiltonian cycle.*

Since we have previously seen conditions that tell us when a graph contains an Eulerian path or an Eulerian circuit, we might assume that there exists another set of conditions that allow us to determine if a graph contains a Hamiltonian path or a Hamiltonian circuit. After all, the only difference between the concepts is that we're considering vertices instead of edges. As it turns out, however, we aren't so lucky; no efficient method is known that solves the problem of determining whether a given graph contains a Hamiltonian path or a Hamiltonian circuit. Recall that Theorem 1.3 gives a simple characterization of graphs with Eulerian circuits.

**Example 1.11** *Consider  $n \geq 1$  and a graph  $G_n = (\{0, 1\}^n, E_n)$  where the edges of  $E_n$  are defined by setting two vertices  $\mathbf{u}$  and  $\mathbf{v}$  ( $\in \{0, 1\}^n$ ) to be adjacent if they differ from each other in exactly one component (that is, the so called Hamming distance of  $\mathbf{u}$  and  $\mathbf{v}$  is one).*

*The graph  $G_n$  is called the  $n$ -cube. The graph  $G_n$  is Hamiltonian for all  $n \geq 1$  and its Hamiltonian cycles correspond to the  $n$ -bit Gray codes of size  $2^n$ . With  $n = 3$  a Hamiltonian cycle of  $G_3$  is*

$$((0, 0, 0), (1, 0, 0), (1, 1, 0), (0, 1, 0), (0, 1, 1), (1, 1, 1), (1, 0, 1), (0, 0, 1), (0, 0, 0))$$

Determining whether a graph is Hamiltonian is a well known algorithmically hard problem (a so call NP-complete problem). However, a problem being algorithmically hard does not prevent us from solving it efficiently in special cases.

There are some basic observations that allow us to conclude immediately that a given graph has a Hamiltonian circuit; for instance,

- every complete graph  $K_n$ ,  $n \geq 3$ , contains a Hamiltonian circuit;
- every cycle graph  $C_n$  contains a Hamiltonian circuit.

Moreover, a pair of theorems exist that relate the existence of Hamiltonian circuits to the degrees of vertices in a graph. These theorems, which we will not prove here, were originally proved by the mathematicians Gabriel Dirac and Øystein Ore. Both theorems essentially state that a graph is Hamiltonian if it contains enough edges.

**Theorem 1.5 (Dirac's theorem)** *If a graph  $G = (V, E)$  with  $n \geq 3$  vertices is such that, for every vertex  $u \in V$ ,  $d(u) \geq n/2$ , then  $G$  contains a Hamiltonian circuit.*

The result by Ore is a strengthening of the result by Dirac.

**Theorem 1.6 (Ore's theorem)** *If a graph  $G = (V, E)$  with  $n \geq 3$  vertices is such that, for every pair of nonadjacent vertices  $u, v \in V$ ,  $d(u) + d(v) \geq n$ , then  $G$  contains a Hamiltonian circuit.*

Keep in mind that all of the conditions we have seen in this section are only sufficient for proving the existence of a Hamiltonian circuit. If some graph does not meet these conditions, then that does not necessarily imply that the graph does not contain a Hamiltonian circuit. Currently, no conditions are known that are both necessary and sufficient for the existence of a Hamiltonian path or Hamiltonian circuit.

### 1.3.1 Applications of Special Paths and Circuits

Earlier we discussed the *traveling salesperson* problem in the context of permutations. The traveling salesperson problem asks, given a list of  $n$  cities and a list of distances between each city, what is the shortest route that both visits all cities and ends in the origin city.

If we model an instance of the traveling salesperson problem as a graph, where vertices are cities and (weighted) edges are distances between cities, then it becomes evident that this

problem is a variant of the Hamiltonian circuit problem. Unfortunately, we already know that no efficient method is known to solve the Hamiltonian circuit problem, so consequently we have no efficient method of solving the traveling salesperson problem either.

So, if we have no efficient method of finding Hamiltonian paths or Hamiltonian circuits, could we instead try to augment a given graph to make it Hamiltonian? This is known as the *Hamiltonian completion* problem, which asks for the minimum number of edges we must add to a graph to ensure it contains a Hamiltonian circuit. Both Dirac's and Ore's theorems give conditions on the number of edges required for a simple graph to contain a Hamiltonian circuit, and the Hamiltonian completion problem has been solved for particular classes of graphs. However, in the general case, we still have no efficient method for determining this minimum number.

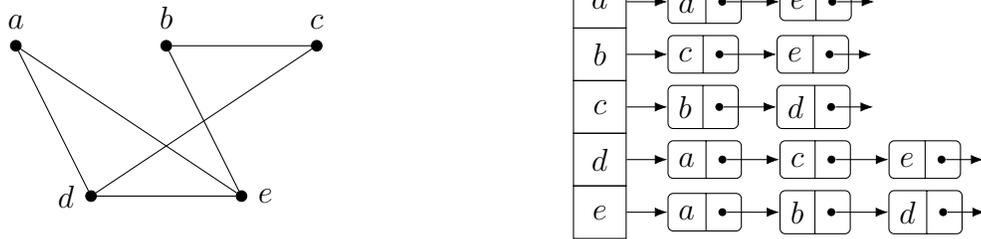
## 1.4 Representing graphs

There are a number of methods of representing graphs apart from listing vertices and edges as explicit sets. Depending on how we are using graphs in an application, representing graphs using such methods could lead to ease-of-use or efficiency gains; for instance, storing vertices and edges in a data structure will result in faster access times compared to storing such data in separate locations on a disk.

The first method of representing graphs is known as an **adjacency list**. This method uses both an array data structure and a linked list data structure: vertices are represented using indices of the array, and for each array entry, the linked list stored at that index records all vertices adjacent to the vertex at that index.

Using adjacency lists, we can access vertices in constant time by indexing into the array, and we may determine all of the vertices adjacent to a given vertex by reading the elements of the linked list, which can be done in time proportional to the degree of the vertex.

**Example 1.12** *The graph at left can be represented by the adjacency list at right.*



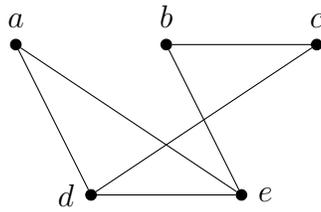
As mentioned previously, a common method of representing graphs is known as an **adjacency matrix**. As the name suggests, this method uses a matrix, or a two-dimensional array, to store vertex and edge data. Given a graph  $G$  with  $n$  vertices, the adjacency matrix of  $G$  (denoted  $A_G$ ) is an  $n \times n$  matrix where the entry at row  $i$  and column  $j$  is equal to 1 when vertices  $u_i$  and  $u_j$  are adjacent and equal to 0 otherwise.

**Remark.** Adjacency matrices can represent loops on a vertex  $u_i$  by storing a 1 entry at position  $(i, i)$  of the matrix. We can modify our implementation of an adjacency matrix to handle graphs with multiple edges by storing the number of edges between vertices  $u_i$  and  $u_j$  at position  $(i, j)$  of the matrix.

As mentioned previously, if the graph  $G$  is undirected, then the associated adjacency matrix  $A_G$  will be symmetric; that is, the entries at positions  $(i, j)$  and  $(j, i)$  will be equal for all  $i$  and  $j$ . This is not always the case if  $G$  is a directed graph.

An adjacency matrix allows us to determine adjacent vertices by scanning a given row or column, which can be done in time proportional to the number of vertices,  $|V|$ . For graphs with many vertices and few edges, adjacency matrices perform worse than adjacency lists. However, adjacency matrices are easier to implement than adjacency lists.

**Example 1.13** *The graph at left (identical to the graph from Example 1.12) can be represented by the adjacency matrix at right.*



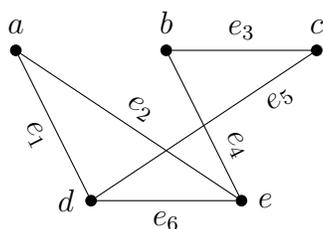
	$a$	$b$	$c$	$d$	$e$
$a$	0	0	0	1	1
$b$	0	0	1	0	1
$c$	0	1	0	1	0
$d$	1	0	1	0	1
$e$	1	1	0	1	0

The third method that we will see in this section also uses a matrix, but in a slightly different way. Instead of using rows and columns to represent vertices in a graph as we did with the adjacency matrix, we represent all vertices as rows and all edges as columns. The resulting structure is known as an **incidence matrix**. Given a graph  $G$  with  $n$  vertices and  $m$  edges, the incidence matrix of  $G$  (denoted  $\mathbf{E}_G$ ) is an  $n \times m$  matrix where the entry at row  $i$  and column  $j$  is equal to 1 if edge  $e_j$  is incident to vertex  $u_i$  and equal to 0 otherwise.

**Remark 1.1** *Just like with adjacency matrices, we can represent loops and multiple edges in an incidence matrix by making the appropriate changes to entries of the matrix. If we want to represent directed edges, we may use positive or negative entries depending on whether an edge is pointing toward or away from a given vertex.*

In order to determine adjacent vertices in an incidence matrix, we must scan both the edge columns (to determine edges incident to our chosen vertex) and the vertex rows (to determine other vertices to which our chosen vertex is adjacent). This can be done in time proportional to the size of the matrix,  $|V| \cdot |E|$ , making it extremely inefficient compared to our previous methods. However, incidence matrices have their advantages; as an example, incidence matrices can distinguish multiple edges, while adjacency matrices cannot.

**Example 1.14** *The graph at left (identical to the graph from Example 1.12 with added edge labels) can be represented by the incidence matrix at right.*



	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$
$a$	1	1	0	0	0	0
$b$	0	0	1	1	0	0
$c$	0	0	1	0	1	0
$d$	1	0	0	0	1	1
$e$	0	1	0	1	0	1

Finally, keep in mind that this section is not exhaustive: there are many other methods to represent graphs, and certain methods lend themselves better to certain graphs. However, the three methods presented here are among the most commonly used methods of graph representation.

## 2 Planarity

We often informally refer to the process of depicting a graph on paper as “drawing”. However, simply saying that we have “drawn” a graph allows some undesirable properties to sneak in that might make our graph confusing to look at. Thus, we want to formalize the notion of drawing in some way to ensure that our depictions of graphs are clear.

Mathematicians have a name for such a formalism: an **embedding** of a graph  $G$  on a surface (for our purposes, a two-dimensional plane) is a representation of the vertices and edges of  $G$  that meet the following criteria:

- each vertex  $u \in V$  is assigned a point on the surface, and no two vertices share the same point;
- each edge  $e \in E$  is assigned a curve on the surface, and no two edges share the same curve;
- the endpoints of some curve  $e$  are exactly the two incident vertices  $u$  and  $v$ ; and
- no vertex other than the two incident vertices  $u$  and  $v$  lie on the edge  $e$ .

Essentially, an embedding ensures that we are able to see all of the vertices and edges of a

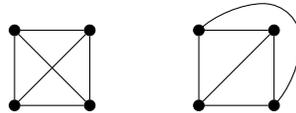
graph and that no vertices or edges overlap one another.

Observe that edges overlapping (that is, one edge covering another edge entirely) is different from edges crossing (that is, one edge intersecting another edge at some point on the surface, which includes an edge intersecting itself). Many graphs have edges that cross, and we have seen such graphs throughout these notes. But a special subset of graphs have edges that do not cross, and that subset is the topic of this section.

**Definition 2.1 (Planar graph)** *A graph  $G = (V, E)$  is planar if  $G$  has an embedding in the plane that is crossing-free.*

Often, if we are given a graph with crossing edges, we can redraw the edges in such a way that no edges cross one another. If we can redraw a graph  $G$  into a crossing-free graph  $G'$ , then we say that  $G'$  is a **planar representation** of  $G$ .

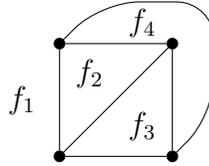
**Example 2.1** *The embedding of the graph  $K_4$  at left is not crossing-free. However, the embedding of the graph at right is both crossing-free and isomorphic to  $K_4$ . Therefore,  $K_4$  is a planar graph.*



In the same way that borders on Earth divide land into countries, the edges of a planar graph divide a surface into **faces**. Faces need not be confined to the interior of a graph; there always exists one exterior, unbounded face that surrounds the graph.

Just like vertices, faces have degrees. The degree of a face is equal to the number of edges making up the boundary of that face. Intuitively, the degree of a face can be thought of as the number of lines you would need to draw, without lifting your pencil, to trace out the face on paper and end up at the point where you started.

**Example 2.2** *Recall the planar embedding of  $K_4$  from Example 2.1. This planar graph consists of four faces, each labelled in the following figure. Observe that one face,  $f_1$ , is an exterior face. Further observe that each face of the graph has degree 3, since each face is bounded by three edges.*



Long ago, Euler discovered a relationship between the numbers of vertices, edges, and faces of certain planar graphs. Summing these numbers in a particular way produces a constant value, known as the **characteristic**.

**Theorem 2.1 (Euler's formula)** *Given a connected planar graph  $G = (V, E)$  with  $v$  vertices,  $e$  edges, and  $f$  faces,*

$$v - e + f = 2.$$

**Proof.** Let  $G$  be a connected planar graph. We will prove the claim using the principle of mathematical induction on the number of edges of  $G$ . Let  $P(e)$  be the statement “given a graph  $G$  with  $v$  vertices and  $e$  edges,  $v - e + f = 2$ ”.

When  $e = 0$ , we know that  $G$  is an edgeless graph. Since  $G$  is connected, it must consist of only one vertex, and as a result,  $G$  contains one exterior face. This gives  $v - e + f = 1 - 0 + 1 = 2$ . Therefore,  $P(0)$  is true.

Assume that  $P(e')$  is true for some  $e' \in \mathbb{N}$ .

We now show that  $P(e' + 1)$  is true; that is, we add one edge to the graph  $G$  and check whether the statement holds. The new edge can be added in one of two ways:

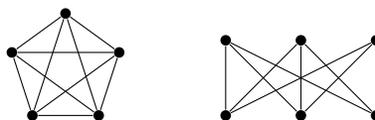
- If the edge is incident to one existing vertex, then we must add a new vertex to  $G$ . This increases both the number of vertices and the number of edges by one, which gives  $(v + 1) - (e' + 1) + f = v + 1 - e' - 1 + f = v - e' + f = 2$ .
- If the edge connects two existing vertices to one another, then we divide some face within  $G$  into two faces. This increases both the number of edges and the number of faces by one, which gives  $v - (e' + 1) + (f + 1) = v - e' - 1 + f + 1 = v - e' + f = 2$ .

In either case,  $P(e' + 1)$  is true. By the principle of mathematical induction,  $P(e)$  is true for all  $e \in \mathbb{N}$ .  $\square$

Note that the fact that  $G$  is connected is crucial to the correctness of Theorem 2.1. Graphs that are not connected have a slightly different formula.

So, why do we care about Euler's formula? Using this result, we can derive a number of conditions that must be met in order for a graph to be planar.

If a graph has no planar representation, then we say that it is **nonplanar**. Proving the nonplanarity of a graph often reduces to showing that the graph does not meet the conditions set out by Euler's formula (or by some corollary of Euler's formula). As illustrative examples of such proofs, we will focus on proving the nonplanarity of two graphs with which we are already familiar:  $K_5$  and  $K_{3,3}$ .



**Theorem 2.2** *The complete graph  $K_5$  is nonplanar.*

**Proof.** Assume by contradiction that  $K_5$  is planar. Then  $K_5$  satisfies Euler's formula. Since the graph consists of 5 vertices and 10 edges, we know that there should exist  $f = 2 - 5 + 10 = 7$  faces in the graph.

Let  $b$  denote the number of “boundary edges” surrounding each face of the graph. Any face must be bounded by at least three edges, so we know that  $b \geq 3f$ . However, since each edge in the graph is a boundary edge for exactly two faces, we also know that  $b = 2e$ . Therefore, we have that  $2e \geq 3f$ .

However, since  $e = 10$  and  $f = 7$ , this inequality asserts that  $20 \geq 21$ , which is impossible. Therefore,  $K_5$  is nonplanar.  $\square$

**Theorem 2.3** *The complete bipartite graph  $K_{3,3}$  is nonplanar.*

**Proof.** Assume by contradiction that  $K_{3,3}$  is planar. Then  $K_{3,3}$  satisfies Euler's formula. Since the graph consists of 6 vertices and 9 edges, we know that there should exist  $f = 2 - 6 + 9 = 5$  faces in the graph.

Again, let  $b$  denote the number of “boundary edges” surrounding each face of the graph. We

know once more that  $b = 2e$ . Since  $K_{3,3}$  is a complete bipartite graph, we also know that  $C_3$  is not a subgraph (by Example 1.7), so each face must be bounded by at least four edges, giving  $b \geq 4f$ .

However, since  $e = 9$  and  $f = 5$ , this inequality asserts that  $18 \geq 20$ , which is impossible. Therefore,  $K_{3,3}$  is nonplanar.  $\square$

At this point, you might wonder why we chose to prove that these two specific graphs were nonplanar. There was a hidden motive for our choices of proofs: we actually require these results to prove the final result of this section. First, however, we require just one more bit of terminology.

It is possible to define an operation on a graph that, given an edge  $e$  incident to vertices  $u$  and  $v$ , divides the edge into two edges  $e_1$  and  $e_2$ , where  $e_1$  is incident to  $u$  and a new vertex  $w$  and  $e_2$  is incident to  $w$  and  $v$ . This operation changes nothing about the graph apart from the number of vertices and edges and, in particular, it does not affect the planarity of the graph. We call this operation a **subdivision**.



Now, we are ready to see a fascinating theorem due to the Polish mathematician Kazimierz Kuratowski. This theorem gives us a pair of conditions that identify when certain graphs are nonplanar. Put simply, if we can obtain a graph  $G$  by taking subdivisions of either of the nonplanar graphs  $K_5$  or  $K_{3,3}$ , then  $G$  itself is nonplanar.

**Theorem 2.4 (Kuratowski's theorem)** *A graph  $G$  is nonplanar if and only if  $G$  contains a subdivision of  $K_5$  or a subdivision of  $K_{3,3}$  as a subgraph.*

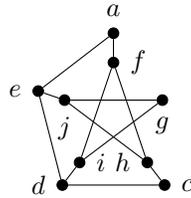
**Proof.** ( $\Rightarrow$ ): Omitted.

( $\Leftarrow$ ): If  $G$  contains a subdivision of  $K_5$  or a subdivision of  $K_{3,3}$  as a subgraph, then  $G$  is nonplanar by Theorem 2.2 or Theorem 2.3, respectively.  $\square$

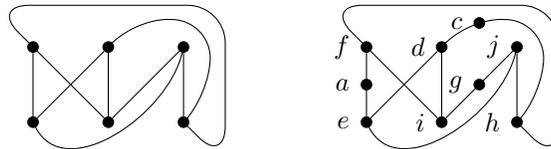
Kuratowski's theorem is biconditional, so we must prove two statements in order to prove the overall theorem. Although one direction of the proof ( $\Leftarrow$ ) followed immediately from

other results we proved, the other direction ( $\Rightarrow$ ) is a long proof and we omit it here, but many textbooks dedicated to graph theory contain the full proof.

**Example 2.3** *We can show that the Petersen graph is nonplanar by using Kuratowski's theorem. First, delete one vertex from the Petersen graph to obtain the following subgraph, then label the vertices.*



*Next, consider the graph at left, which is isomorphic to the complete bipartite graph  $K_{3,3}$ . Take a subdivision of this graph to obtain the graph at right, and again label the vertices.*



*Finally, observe that the given labelings establish an isomorphism between the two graphs, meaning that the Petersen graph contains a subdivision of  $K_{3,3}$  as a subgraph. By Kuratowski's theorem, the Petersen graph is nonplanar.*

### 3 Graph colourings

In this section, we will be doing something that we don't usually get to do in a university-level course: colouring. (Hopefully you remembered to bring your crayons!)

In some of the graphs we have seen, we labelled vertices with symbols like  $a$ ,  $b$ ,  $u$ , and  $v$ . Importantly, each of these symbols had to be unique, since we wanted to distinguish between vertices. A *graph colouring*, on the other hand, is like a method of nonunique labelling. We can colour any aspect of a graph, such as vertices, edges, or faces.

**Remark.** For the remainder of this section, we will focus on vertex colourings. Similar definitions and results exist for both edge colourings and face colourings, but we will not

discuss either topic here.

Instead of giving each vertex a symbolic label, we can assign a “colour” to each vertex that distinguishes it from nearby vertices. These colours need not be unique; we can label more than one vertex with the same colour.

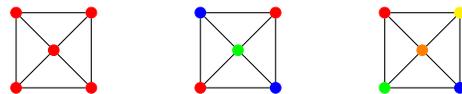
In our formal definition, we use numerical values to represent colours.

**Definition 3.1 (Vertex colouring)** *Given a graph  $G = (V, E)$  and a set of  $k$  colours  $\{1, 2, \dots, k\}$ , a vertex colouring of  $G$  is a function  $f : V \rightarrow \{1, 2, \dots, k\}$ .*

If a vertex colouring  $f$  has the property that, for all edges  $\{u, v\} \in E$ ,  $f(u) \neq f(v)$ , then we say that the vertex colouring is **proper**. A proper vertex colouring is one in which no adjacent vertices share the same colour.

If we can construct a proper vertex colouring of a graph using at most  $k$  colours, then we say that the graph is  **$k$ -colourable**.

**Example 3.1** *Each of the following graphs depicts a vertex colouring. However, only the two rightmost graphs depict a proper vertex colouring. Since we can obtain a proper vertex colouring using three colours, this graph is 3-colourable.*



### 3.1 Chromatic Numbers

The question we typically want to answer is how many colours are required in order to construct a proper vertex colouring of a graph  $G$ . This idea leads to the notion of the *chromatic number* of  $G$ .

**Definition 3.2 (Chromatic number)** *Given a graph  $G$ , the chromatic number of  $G$ , denoted by  $\chi(G)$ , is equal to the smallest number of colours required to construct a proper vertex colouring of  $G$ .*

We can attain both the upper bound and the lower bound on the chromatic number if we

consider particular classes of graphs.

- Given a complete graph  $K_n$  with  $n$  vertices and  $\frac{n(n-1)}{2}$  edges, we have that  $\chi(K_n) = n$ . This is because each vertex is adjacent to every other vertex in the graph.
- Given an edgeless graph  $0_n$  with  $n$  vertices and zero edges, we have that  $\chi(0_n) = 1$ . This is because no vertices are adjacent to one another in the graph.

Thus, we know that for any graph  $G$  with  $n$  vertices, we have that  $1 \leq \chi(G) \leq n$ .

With a little work, we can obtain a few more results about the chromatic numbers of given graphs. Our first result relates the chromatic number of a graph to the *maximum degree* of any vertex in the graph, denoted by  $\Delta$ .

**Theorem 3.1** *If a graph  $G$  has maximum degree  $\Delta$ , then  $\chi(G) \leq \Delta + 1$ .*

**Proof.** Assume we have a set of  $\Delta + 1$  colours. Choose an arbitrary vertex, say  $v_1$ , from the graph  $G$  and assign to it a colour. Repeat this process of assigning colours to arbitrary vertices of  $G$  under the condition that, if two vertices  $v_i$  and  $v_j$  are adjacent, then both vertices should be assigned different colours.

Since the maximum degree of  $G$  is  $\Delta$  and since we have  $\Delta + 1$  colours available to use, we will not encounter the issue of running out of colours to assign to adjacent vertices. Therefore,  $\chi(G) \leq \Delta + 1$ .  $\square$

A stronger result, proved by the English mathematician R. Leonard Brooks, reduces the bound given in Theorem 3.1 to  $\chi(G) \leq \Delta$ , unless  $G = K_n$  or  $G = C_{2n+1}$ . However, Brooks' result is harder to prove.

Our next result relates the chromatic number of a graph to the chromatic numbers of its subgraphs.

**Theorem 3.2** *If a graph  $G$  contains a subgraph  $H$ , then  $\chi(H) \leq \chi(G)$ .*

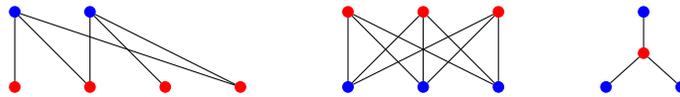
**Proof.** Given a proper vertex colouring of the graph  $G$ , we can copy those colours to the corresponding vertices of  $H$ . Thus, a proper vertex colouring of  $H$  requires at most  $\chi(G)$  colours.  $\square$

Next, recall the notion of a bipartite graph from our discussion on special graph classes. A bipartite graph is a graph whose vertex set  $V$  can be partitioned into two subsets  $V_1$  and  $V_2$  such that each edge joins a vertex in  $V_1$  to a vertex in  $V_2$ . The following result characterizes the set of bipartite graphs in terms of their chromatic number.

**Theorem 3.3** *A graph  $G$  is a bipartite graph if and only if  $\chi(G) = 2$ .*

**Proof.** ( $\Rightarrow$ ): Given a bipartite graph  $G$ , colour each of the vertices in  $V_1$  one colour (say, red), and colour each of the vertices in  $V_2$  another colour (say, blue). No vertices in  $V_1$  are adjacent to vertices in the same set, and the same is true for vertices in  $V_2$ . Thus,  $\chi(G) = 2$ .

( $\Leftarrow$ ): Given a graph  $G$  with  $\chi(G) = 2$ , partition each of the vertices of the first colour into one set  $V_1$ , and partition each of the vertices of the second colour into another set  $V_2$ . Since no two adjacent vertices of  $G$  share the same colour, vertices in  $V_1$  are only adjacent to vertices in  $V_2$  and vice versa, making the graph bipartite.  $\square$



Finally, let's investigate chromatic numbers of planar graphs. We do so in the context of maps: a paper map is a plane, land areas are vertices of a graph, and adjacent land areas are connected by non-crossing edges.

In the early days of graph theory, a South African mathematician named Francis Guthrie made a conjecture while trying to colour a map of England. Guthrie asserted that one needs only four colours to colour in any map, no matter how many regions the map depicts. In 1879, an English mathematician named Alfred Kempe claimed to have a proof of Guthrie's conjecture, and for this he received many awards and accolades. However, 11 years later, another English mathematician named Percy Heawood found a flaw in Kempe's proof. Heawood gave his own proof that five colours are in fact sufficient.

After the late 19th century work on map colouring lay dormant for many years. Then, in 1976, Kenneth Appel and Wolfgang Haken proved that, indeed, four colours are sufficient to colour any map. The proof of Appel and Haken was a landmark result, since it was one of the first major mathematical theorems to be proved with the aid of a computer; the

complete proof consisted of 1 936 different cases. Since then, the proof has been shortened considerably—to a mere 633 cases—but it is still too lengthy to reproduce here.

**Theorem 3.4 (Four colour theorem)** *If a graph  $G$  is planar, then  $\chi(G) \leq 4$ .*