Chapter 10

Graphs

A **graph** is a structure in which pairs of **vertices** are connected by **edges**. Each edge may act like an ordered pair (in a **directed graph**) or an unordered pair (in an **undirected graph**). We've already seen directed graphs as a representation for relations. Most work in graph theory concentrates instead on undirected graphs.

Because graph theory has been studied for many centuries in many languages, it has accumulated a bewildering variety of terminology, with multiple terms for the same concept (*e*.g. **node** for vertex or **arc** for edge) and ambiguous definitions of certain terms (*e*.g., a "graph" without qualification might be either a directed or undirected graph, depending on who is using the term: graph theorists tend to mean undirected graphs, but you can't always tell without looking at the context). We will try to stick with consistent terminology to the extent that we can. In particular, unless otherwise specified, a *graph* will refer to a **finite simple undirected graph**: an undirected graph with a finite number of vertices, where each edge connects two distinct vertices (thus no **self-loops**) and there is at most one edge between each pair of vertices (no **parallel edges**).

A reasonably complete glossary of graph theory can be found at at http://en.wikipedia.org/wiki/Glossary_of_graph_theory. See also Ferland [Fer08], Chapters 8 and 9; Rosen [Ros12] Chapter 10; or Biggs [Big02] Chapter 15 (for undirected graphs) and 18 (for directed graphs).

If you want to get a fuller sense of the scope of graph theory, Reinhard Diestel's (graduate) textbook *Graph Theory*[Die10] can be downloaded from http://diestel-graph-theory.com.

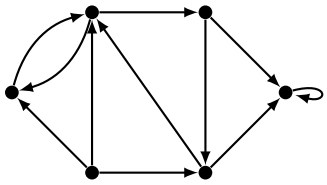


Figure 10.1: A directed graph

10.1 Types of graphs

Graphs are represented as ordered pairs $G = (V, E)$, where V is a set of vertices and E a set of edges. The differences between different types of graphs depends on what can go in E. When not otherwise specified, we usually think of a *graph* as an *undirected graph* (see below), but there are other variants. Typically we assume that V and E are both finite.

10.1.1 Directed graphs

In a **directed graph** or *digraph*, each element of E is an ordered pair, and we think of edges as arrows from a **source**, **head**, or **initial vertex** to a **sink**, **tail**, or **terminal vertex**; each of these two vertices is called an **endpoint** of the edge. A directed graph is **simple** if there is at most one edge from one vertex to another. A directed graph that has multiple edges from some vertex u to some other vertex v is called a **directed multigraph**.

For simple directed graphs, we can save a lot of ink by adopting the convention of writing an edge (u, v) from u to v as just uv.

Directed graphs are drawn as in Figure 10.1.

As we saw in the notes on relations, there is a one-to-one correspondence between simple directed graphs with vertex set V and relations on V.

10.1.2 Undirected graphs

In an **undirected graph**, each edge is an undirected pair, which we can represent as subset of V with one or two elements. A **simple undirected graph** contains no duplicate edges and no **loops** (an edge from some vertex u back to itself); this means we can represent all edges as two-element subsets of V. Most of the time, when we say *graph*, we mean a simple undirected graph. Though it is possible to consider infinite graphs, for convenience we will limit ourselves to finite graphs, where $n = |V|$ and $m = |E|$ are both natural numbers.

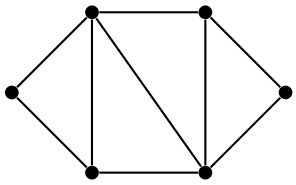


Figure 10.2: A graph

As with directed graphs, instead of writing an edge as $\{u, v\}$, we will write an edge between u and v as just uv. Note that in an undirected graph, uv and vu are the same edge.

Graphs are drawn just like directed graphs, except that the edges don't have arrowheads on them. See Figure 10.2 for an example.

If we have loops or parallel edges, we have a more complicated structure called a **multigraph**. This requires a different representation where elements of E are abstract edges and we have a function mapping each element of E to its endpoints. Some authors make a distinction between *pseudographs* (with loops) and *multigraphs* (without loops), but we'll use multigraph for both.

Simple undirected graphs also correspond to relations, with the restriction that the relation must be irreflexive (no loops) and symmetric (undirected edges). This also gives a representation of undirected graphs as directed graphs, where the edges of the directed graph always appear in pairs going in opposite directions.

10.1.3 Hypergraphs

In a **hypergraph**, the edges (called **hyperedges**) are arbitrary nonempty sets of vertices. A k-**hypergraph** is one in which all such hyperedges connected exactly k vertices; an ordinary graph is thus a 2-hypergraph.

Hypergraphs can be drawn by representing each hyperedge as a closed curve containing its members, as in the left-hand side of Figure 10.3.

Hypergraphs aren't used very much, because it is always possible (though not always convenient) to represent a hypergraph by a **bipartite graph**. In a bipartite graph, the vertex set can be partitioned into two subsets S and T, such that every edge connects a vertex in S with a vertex in T. To represent a hypergraph H as a bipartite graph, we simply represent the vertices of H as vertices in S and the hyperedges of H as vertices in T, and put in an edge (s, t) whenever s is a member of the hyperedge t in H. The right-hand side of Figure 10.3 gives an example.

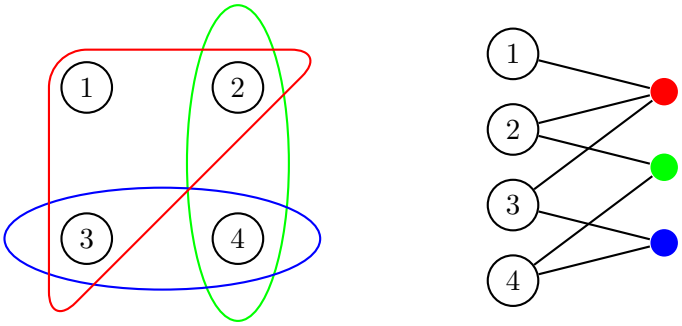
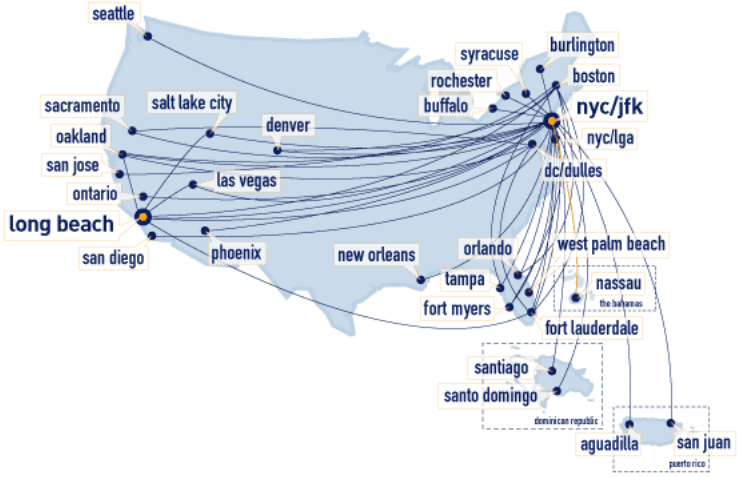


Figure 10.3: Two representations of a hypergraph. On the left, four vertices are connected by three hyperedges. On the right, the same four vertices are connected by ordinary edges to new vertices representing the hyperedges.

10.2 Examples of graphs

Any relation produces a graph, which is directed for an arbitrary relation and undirected for a symmetric relation. Examples are graphs of parenthood (directed), siblinghood (undirected), handshakes (undirected), etc.

Graphs often arise in transportation and communication networks. Here's a (now very out-of-date) route map for Jet Blue airlines, originally taken from http://www.jetblue.com/travelinfo/routemap.html:



Such graphs are often **labeled** with edge lengths, prices, etc. In computer networking, the design of network graphs that permit efficient routing of data without congestion, roundabout paths, or excessively large routing tables is

a central problem.

The **web graph** is a directed multigraph with web pages for vertices and hyperlinks for edges. Though it changes constantly, its properties have been fanatically studied both by academic graph theorists and employees of search engine companies, many of which are still in business. Companies like Google base their search rankings largely on structural properties of the web graph.

Peer-to-peer systems for data sharing often have a graph structure, where each peer is a node and connections between peers are edges. The problem of designing efficient peer-to-peer systems is similar in many ways to the problem of designing efficient networks; in both cases, the structure (or lack thereof) of the underlying graph strongly affects efficiency.

10.3 Local structure of graphs

There are some useful standard terms for describing the immediate connections of vertices and edges:

- Incidence: a vertex is **incident** to any edge of which it is an endpoint (and vice versa).
- Adjacency, neighborhood: two vertices are **adjacent** if they are the endpoints of some edge. The **neighborhood** of a vertex v is the set of all vertices that are adjacent to v.
- Degree, in-degree, out-degree: the **degree** of v counts the number edges incident to v. In a directed graph, **in-degree** counts only incoming edges and **out-degree** counts only outgoing edges (so that the degree is always the in-degree plus the out-degree). The degree of a vertex v is often abbreviated as $d(v)$; in-degree and out-degree are similarly abbreviated as $d^-(v)$ and $d^+(v)$, respectively.

10.4 Some standard graphs

Most graphs have no particular structure, but there are some families of graphs for which it is convenient to have standard names. Some examples are:

- **Complete graph** K_n. This has n vertices, and every pair of vertices has an edge between them. See Figure 10.4.

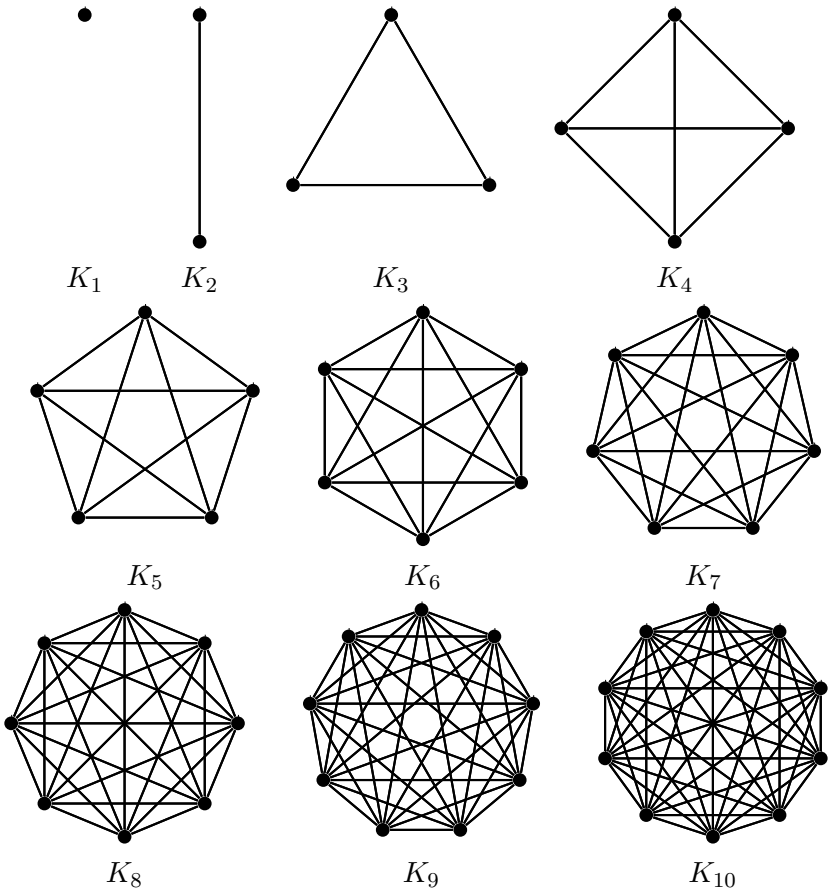


Figure 10.4: Complete graphs K_1 through K_{10}

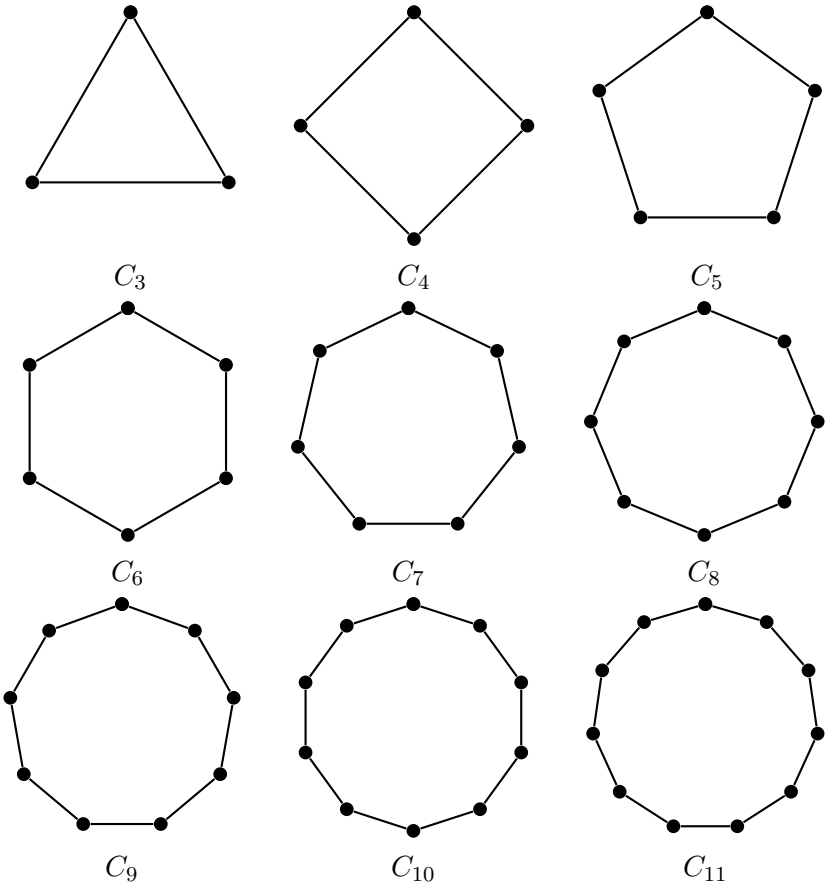


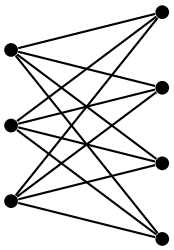
Figure 10.5: Cycle graphs C_3 through C_{11}

- **Cycle** graph C_n. This has vertices $\{0, 1, \ldots n-1\}$ and an edge from i to $i+1$ for each i, plus an edge from $n-1$ to 0. For any cycle, n must be at least 3. See Figure 10.5.

- **Path** P_n. This has vertices $\{0, 1, 2, \ldots n\}$ and an edge from i to $i+1$ for each i. Note that, despite the usual convention, n counts the number of *edges* rather than the number of vertices; we call the number of edges the **length** of the path. See Figure 10.6.

- **Complete bipartite graph** $K_{m,n}$. This has a set A of m vertices and a set B of n vertices, with an edge between every vertex in A and every vertex in B, but no edges within A or B. See Figure 10.7.

Figure 10.6: Path graphs P_0 through P_4



$K_{3,4}$

Figure 10.7: Complete bipartite graph $K_{3,4}$

- **Star graphs**. These have a single central vertex that is connected to n outer vertices, and are the same as $K_{1,n}$. See Figure 10.8.

- The **cube** Q_n. This is defined by letting the vertex set consist of all n-bit strings, and putting an edge between u and u' if u and u' differ in exactly one place. It can also be defined by taking the n-fold square product of an edge with itself (see §10.6).

- **Cayley graphs**. The Cayley graph of a group G with a given set of generators S is a labeled directed graph. The vertices of this graph are the group elements, and for each element g in G and generator s in S there is a directed edge from g to gs labeled with s. An example of a small Cayley graph, based on the **dihedral group** D_4 of symmetries of the square, is given in Figure 10.9.

  Many common graphs are Cayley graphs with the labels (and possibly edge orientations) removed; for example, a directed cycle on m elements is the Cayley graph of $\mathbb{Z}_m$ with generator 1, an $n \times m$ torus is the Cayley graph of $\mathbb{Z}_n \times \mathbb{Z}_m$ with generators $(1, 0)$ and $(0, 1)$, and the cube Q_n is the Cayley graph of $(\mathbb{Z}_2)^n$ with generators all vectors that are zero in all positions but one.

Graphs may not always be drawn in a way that makes their structure obvious. For example, Figure 10.10 shows two different presentations of Q_3, neither of which looks much like the other.

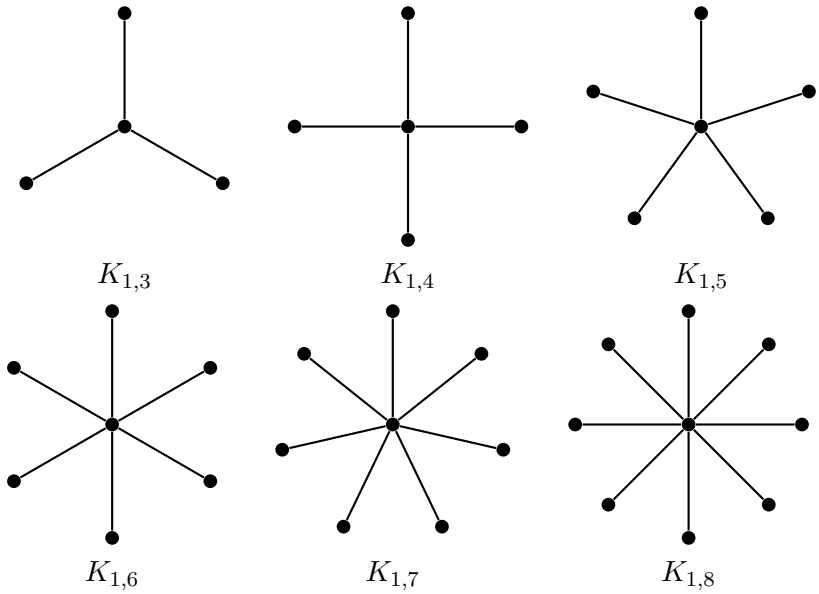


Figure 10.8: star graphs $K_{1,3}$ through $K_{1,8}$

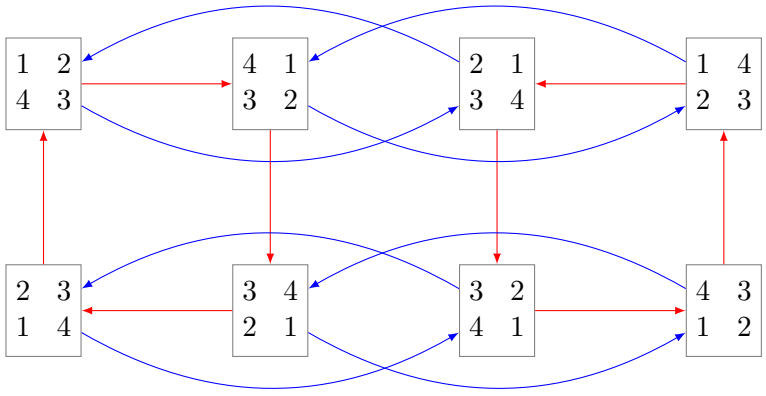


Figure 10.9: Cayley graph of the dihedral group D_4, with generators a corresponding to a clockwise rotation (red arrows) and b corresponding to a flip around the vertical axis (blue arrows). Note that this is a directed graph.

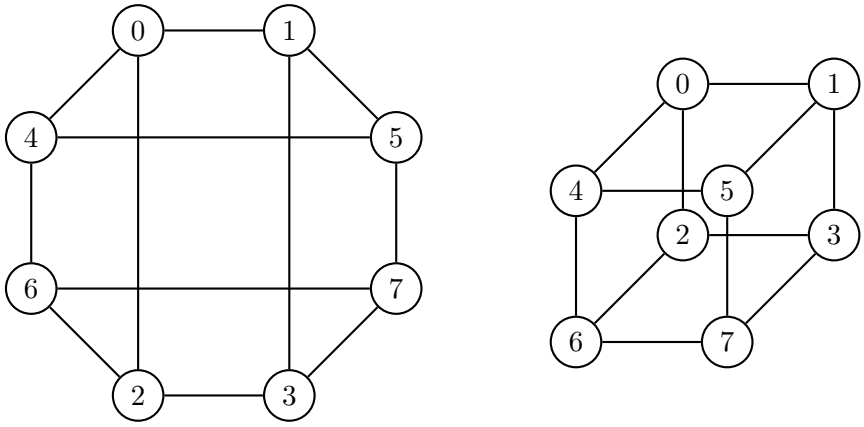


Figure 10.10: Two presentations of the cube graph Q_3

10.5 Subgraphs and minors

A graph G is a **subgraph** of of a graph H, written $G \subseteq H$, if $V_G \subseteq V_H$ and $E_G \subseteq E_H$. We will also sometimes say that G is a subgraph of H if it is isomorphic to a subgraph of H, which is equivalent to having an injective homomorphism from G to H.

One can get a subgraph by deleting edges or vertices or both. Note that deleting a vertex also requires deleting any edges incident to the vertex (since we can't have an edge with a missing endpoint). If we delete as few edges as possible, we get an **induced subgraph**. Formally, the subgraph of a graph H whose vertex set is S and that contains every edge in H with endpoints in S is called the subgraph of H induced by S.

A **minor** of a graph H is a graph obtained from H by deleting edges and/or vertices (as in a subgraph) and **contracting** edges, where two adjacent vertices u and v are merged together into a single vertex that is adjacent to all of the previous neighbors of both vertices. Minors are useful for recognizing certain classes of graphs. For example, a graph can be drawn in the plane without any crossing edges if and only if it doesn't contain K_5 or $K_{3,3}$ as a minor (this is known as **Wagner's theorem**).

Figure 10.11 shows some subgraphs and minors of the graph from Figure 10.2.

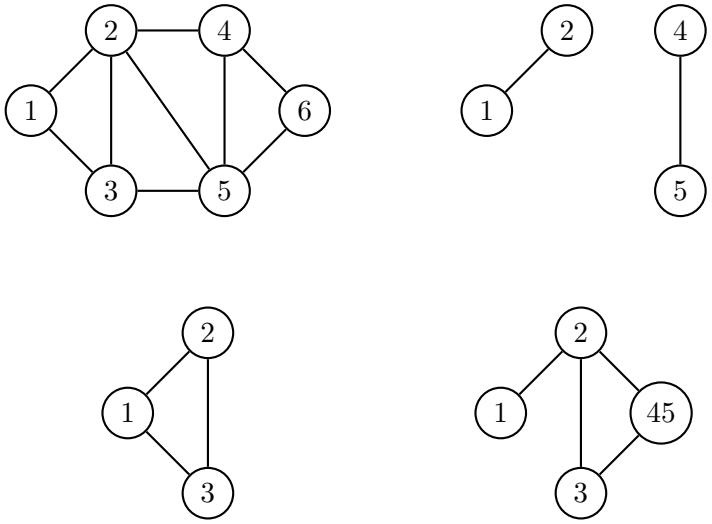


Figure 10.11: Examples of subgraphs and minors. Top left is the original graph. Top right is a subgraph that is not an induced subgraph. Bottom left is an induced subgraph. Bottom right is a minor.

10.6 Graph products

There are at least five different definitions of the product of two graphs used by serious graph theorists. In each case the vertex set of the product is the Cartesian product of the vertex sets, but the different definitions throw in different sets of edges. Two of them are used most often:

- The **square product** or **graph Cartesian product** $G \square H$. An edge $(u, u')(v, v')$ is in $G \square H$ if and only if (a) $u = v$ and $u'v'$ is an edge in H, or (b) uv is an edge in G and $u' = v'$. It's called the square product because the product of two (undirected) edges looks like a square. The intuition is that each vertex in G is replaced by a copy of H, and then corresponding vertices in the different copies of H are linked whenever the original vertices in G are adjacent. For algebraists, square products are popular because they behave correctly for Cayley graphs: if C_1 and C_2 are the Cayley graphs of G_1 and G_2 (for particular choices of generators), then $C_1 \square C_2$ is the Cayley graph of $G_1 \times G_2$.
  - The cube Q_n can be defined recursively by $Q_1 = P_1$ and $Q_n = Q_{n-1} \square Q_1$. It is also the case that $Q_n = Q_k \square Q_{n-k}$.
  - An n-by-m **mesh** is given by $P_{n-1} \square P_{m-1}$.

- The **cross product** or **categorical graph product** $G \times H$. Now $(u, u')(v, v')$ is in $G \times H$ if and only if uv is in G and $u'v'$ is in H. In the cross product, the product of two (again undirected) edges is a cross: an edge from (u, u') to (v, v') and one from (u, v') to (v, u'). The cross product is not as useful as the square product for defining nice-looking graphs, but it can arise in some other situations. An example is when G and H describe the positions (vertices) and moves (directed edges) of two solitaire games; then the cross product $G \times H$ describes the combined game in which at each step the player must make a move in both games. (In contrast, the square product $G \square H$ describes a game where the player can choose at each step to make a move in either game.)

10.7 Functions between graphs

A function from a graph G to another graph H typically maps V_G to V_H, with the edges coming along for the ride. For simplicity, we will generally write $f : G \to H$ when we really mean $f : V_G \to V_H$.

A function $f : G \to H$ is a **graph homomorphism** if, for every edge uv in G, $f(u)f(v)$ is an edge in H. Note that this only goes one way: it is possible to have an edge $f(u)f(v)$ in H but no edge uv in G. Generally we will only be interested in functions between graphs that are homomorphisms, and even among homomorphisms, some functions are more interesting than others.

A graph homomorphism that has an inverse that is also a graph homomorphism is called an **graph isomorphism**. Two graphs G and H are **isomorphic** if there is an isomorphism between them. Intuitively, this means that G and H are basically the same graph, with different names for the vertices, and we will often treat them as the same graph. So, for example, we will think of a graph $G = (V, E)$ where $V = \{1, 3, 5\}$ and $E = \{\{1, 3\}, \{3, 5\}, \{1, 5\}\}$ as an instance of C_3 and K_3 even if the vertex labels are not what we might have chosen by default. To avoid confusion with set equality, we write $G \cong H$ when G and H are isomorphic.

Every graph is isomorphic to itself, because the identity function is an isomorphism. But some graphs have additional isomorphisms. An isomorphism from G to G is called an **automorphism** of G and corresponds to an internal symmetry of G. For example, the cycle C_n has $2n$ different automorphisms (to count them, observe there are n places we can send vertex 0 to, and having picked a place to send vertex 0 to, there are only 2

places to send vertex 1; so we have essentially n rotations times 2 for flipping or not flipping the graph). A path P_n (when $n > 1$) has 2 automorphisms (reverse the direction or not). Many graphs have no automorphisms except the identity map.

An injective homomorphism from G to H is an isormophism between G and some subgraph H' of H. In this case, we often say that G is a subgraph of H, even though technically it is just a copy of G that appears as a subgraph of H. This allows us to say, for example, that P_n is a subgraph of P_{n+1},[1] or all graphs on at most n vertices are subgraphs of K_n.

Homomorphisms that are not injective are not as useful, but they can can sometimes be used to characterize particular classes of graphs indirectly. For example, There is a homomorphism from a graph G to P_1 if and only if G is bipartite (see §C.7.2 for a proof). In general, there is a homomorphism from G to K_n if and only if G is n-partite (recall $P_1 \cong K_2$).

10.8 Paths and connectivity

A fundamental property of graphs is connectivity: whether the graph can be divided into two or more pieces with no edges between them. Often it makes sense to talk about this in terms of reachability, or whether you can get from one vertex to another along some **path**.

The pedantic definition of a path **path** of **length** n in a graph is the image of a homomorphism from P_n. In ordinary speech, it's a sequence of $n + 1$ vertices $v_0, v_1, \ldots, v_n$ such that $v_i v_{i+1}$ is an edge in the graph for each i. A path is **simple** if the same vertex never appears twice (i.e. if the homomorphism is injective). If there is a path from u to v, there is a simple path from u to v obtained by removing cycles (Lemma 10.10.1).

If there is a path from u to v, then v is **reachable** from u: $u \xrightarrow{*} v$. We also say that u is **connected to** v. It's easy to see that connectivity is reflexive (take a path of length 0) and transitive (paste a path from u to v together with a path from v to w to get a path from u to w). But it's not necessarily symmetric if we have a directed graph.

In an undirected graph, connectivity *is* symmetric, so it's an equivalence relation. The equivalence classes of $\xrightarrow{*}$ are called the **connected components** of G, and G itself is **connected** if and only if it has a single connected component, i.e., if every vertex is reachable from every other vertex. (Note that isolated vertices count as (separate) connected components.)

[1] In four different ways!

In a directed graph, we can make connectivity symmetric in one of two different ways:

- Define u to be **strongly connected** to v if $u \xrightarrow{*} v$ and $v \xrightarrow{*} u$. I.e., u and v are strongly connected if you can go from u to v and back again (not necessarily through the same vertices).

  It's easy to see that strong connectivity is an equivalence relation. The equivalence classes are called **strongly-connected components**. A graph G is **strongly connected** if it has one strongly-connected component, i.e., if every vertex is reachable from every other vertex.

- Define u to be **weakly connected** to v if $u \xrightarrow{*} v$ in the undirected graph obtained by ignoring edge orientation. The intuition is that u is weakly connected to v if there is a path from u to v if you are allowed to cross edges backwards. Weakly-connected components are defined by equivalence classes; a graph is weakly-connected if it has one component. Weak connectivity is a "weaker" property that strong connectivity in the sense that if u is strongly connected to v, then u is weakly connected to v; but the converse does not necessarily hold.

The k-th **power** G^k of a graph G has the same vertices as G, but uv is an edge in G^k if and only if there is a path of length k from u to v in G. The **transitive closure** of a directed graph: $G^* = \bigcup_{k=0}^{\infty} G^k$. I.e., there is an edge uv in G^* if and only if there is a path (of any length, including zero) from u to v in G, or in other words if $u \xrightarrow{*} v$. This is equivalent to taking the transitive closure of the adjacency relation.

10.9 Cycles

The standard cycle graph C_n has vertices $\{0, 1, \ldots, n-1\}$ with an edge from i to $i+1$ for each i and from $n-1$ to 0. To avoid degeneracies, n must be at least 3. A **simple cycle** of length n in a graph G is an embedding of C_n in G: this means a sequence of distinct vertices $v_0 v_1 v_2 \ldots v_{n-1}$, where each pair $v_i v_{i+1}$ is an edge in G, as well as $v_{n-1} v_0$. If we omit the requirement that the vertices are distinct, but insist on distinct edges instead, we have a **cycle**. If we omit both requirements, we get a **closed walk**; this includes very non-cyclic-looking walks like the short excursion uvu. We will mostly worry about cycles.[2] See Figure 10.12

[2] Some authors reserve *cycle* for what we are calling a *simple cycle*, and use *circuit* for *cycle*.

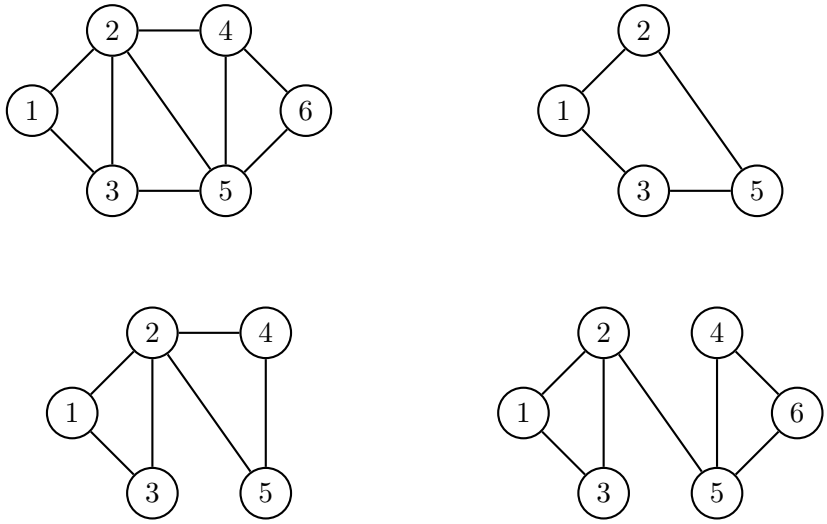


Figure 10.12: Examples of cycles and closed walks. Top left is a graph. Top right shows the simple cycle 1253 found in this graph. Bottom left shows the cycle 124523, which is not simple. Bottom right shows the closed walk 12546523, which uses the 25 edge twice.

Unlike paths, which have endpoints, no vertex in a cycle has a special role.

A graph with no cycles is **acyclic**. **Directed acyclic graphs** or **DAGs** have the property that their reachability relation $\xrightarrow{*}$ is a partial order; this is easily proven by showing that if $\xrightarrow{*}$ is not anti-symmetric, then there is a cycle consisting of the paths between two non-anti-symmetric vertices $u \xrightarrow{*} v$ and $v \xrightarrow{*} u$. Directed acyclic graphs may also be **topologically sorted**: their vertices ordered as $v_0, v_1, \ldots, v_{n-1}$, so that if there is an edge from v_i to v_j, then $i < j$. The proof is by induction on $|V|$, with the induction step setting v_{n-1} to equal some vertex with out-degree 0 and ordering the remaining vertices recursively. (See §9.5.5.1.)

Connected acyclic undirected graphs are called **trees**. A connected graph $G = (V, E)$ is a tree if and only if $|E| = |V| - 1$; we'll prove this and other characterizations of tree in §10.10.3.

A cycle that includes every edge exactly once is called an **Eulerian cycle** or **Eulerian tour**, after Leonhard Euler, whose study of the *Seven bridges of Königsberg* problem led to the development of graph theory. A cycle that includes every vertex exactly once is called a **Hamiltonian cycle** or

Hamiltonian tour, after William Rowan Hamilton, another historical graph-theory heavyweight (although he is more famous for inventing quaternions and the Hamiltonian). Graphs with Eulerian cycles have a simple characterization: a graph has an Eulerian cycle if and only if every vertex has even degree. Graphs with Hamiltonian cycles are harder to recognize.

10.10 Proving things about graphs

Suppose we want to show that all graphs or perhaps all graphs satisfying certain criteria have some property. How do we do this? In the ideal case, we can decompose the graph into pieces somehow and use induction on the number of vertices or the number of edges. If this doesn't work, we may have to look for some properties of the graph we can exploit to construct an explicit proof of what we want.

10.10.1 Paths and simple paths

If all we care about is connectivity, we can avoid making distinctions between paths and simple paths.

Lemma 10.10.1. *If there is a path from s to t in G, there is a simple path from s to t in G.*

Proof. By induction on the length of the path. Specifically, we will show that if there is a path from s to t of length k, there is a simple path from s to t.

The base case is when $k = 1$; then the path consists of exactly one edge and is simple.

For larger k, let $s = v_0 \dots v_k = t$ be a path in G. If this path is simple, we are done. Otherwise, there exist positions $i < j$ such that $v_i = v_j$. Construct a new path $v_1 \dots v_i v_{j+1} \dots v_k$; this is an s–t path of length less than k, so by the induction hypothesis a simple s–t path exists. □

The converse of this lemma is trivial: any simple path is also a path.

Essentially the same argument works for cycles:

Lemma 10.10.2. *If there is a cycle in G, there is a simple cycle in G.*

Proof. As in the previous lemma, we prove that there exists a simple cycle if there is a cycle of length k for any k, by induction on k. First observe that the smallest possible cycle has length 3, since anything shorter either doesn't get back to its starting point or violates the no-duplicate edges

requirement. So the base case is $k = 3$, and it's easy to see that all 3-cycles are simple. For larger k, if $v_0 v_1 \dots v_{k-1}$ is a k-cycle that is not simple, there exist $i < j$ with $v_i = v_j$; patch the edges between them out to get a smaller cycle $v_0 \dots v_i v_{j+1} \dots v_{k-1}$. The induction hypothesis does the rest of the work. □

10.10.2 The Handshaking Lemma

This lemma relates the total degree of a graph to the number of edges. The intuition is that each edge adds one to the degree of both of its endpoints, so the total degree of all vertices is twice the number of edges.

Lemma 10.10.3. *For any graph $G = (V, E)$,*

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

Proof. By induction on $m = |E|$. If $m = 0$, G has no edges, and $\sum_{v \in V} d(v) = \sum_{v \in V} 0 = 0 = 2m$. If $m > 0$, choose some edge st and let $G' = G - st$ be the subgraph of G obtained by removing st. Applying the induction hypothesis to G',

$$\begin{aligned} 2(m-1) &= \sum_{v \in V} d_{G'}(v) \\ &= \sum_{v \in V \setminus \{s,t\}} d_{G'}(v) + d_{G'}(s) + d_{G'}(t) \\ &= \sum_{v \in V \setminus \{s,t\}} d_G(v) + (d_G(s) - 1) + (d_G(t) - 1) \\ &= \sum_{v \in V} d_G(v) - 2. \end{aligned}$$

So $\sum_{v \in V} d_G(v) - 2 = 2m - 2$, giving $\sum_{v \in V} d_G(v) = 2m$. □

One application of the lemma is that the number of odd-degree vertices in a graph is always even (take both sides mod 2). Another, that we'll use below, is that if a graph has very few edges, then it must have some low-degree vertices.

10.10.3 Characterizations of trees

A **tree** is defined to be an acyclic connected graph. There are several equivalent characterizations.

Theorem 10.10.4. *A graph is a tree if and only if there is exactly one simple path between any two distinct vertices.*

Proof. A graph G is connected if and only if there is *at least* one simple path between any two distinct vertices. We'll show that it is acyclic if and only if there is *at most* one simple path between any two distinct vertices.

First, suppose that G has two distinct simple paths $u = v_1 v_2 \dots v_k = v$ and $u = v'_1 v'_2 \dots v'_\ell = v$. Let i be the largest index for which $v_i = v'_i$; under the assumption that the paths are distinct and simple , we have $i < \min(k, \ell)$. Let $j > i$ be the smallest index for which $v_j = v'_m$ for some $m > i$; we know that some such j exists because, if nothing else, $v_k = v_\ell$. Let m is the smallest such m.

Now construct a cycle $v_i v_{i+1} \dots v_j v'_{m-1} v'_{m-2} \dots v'_i = v_i$. This is in fact a simple cycle, since the v_r are all distinct, the v'_s are all distinct, and if any v_r with $i < r < j$ equals v'_s with $i < s < m$, then j or m is not minimal. It follows that if G has two distinct simple paths between the same vertices, it contains a simple cycle, and is not acyclic.

Conversely, suppose that G is not acyclic, and let $v_1 v_2 \dots v_k = v_1$ be a simple cycle in G. Then $v_1 v_2$ and $v_2 \dots v_k$ are both simple paths between v_1 and v_2, one of which contains v_3 and one of which doesn't. So if G is not acyclic, it contains more than one simple path between some pair of vertices. □

An alternative characterization counts the number of edges: we will show that any graph with less than $|V| - 1$ edges is disconnected, and any graph with more than $|V| - 1$ edges is cyclic. With exactly $|V| - 1$ edges, we will show that a graph is connected if and only if it is acyclic.

The main trick involves reducing a $|V|$ by removing a degree-1 vertex. The following lemma shows that this does not change whether or not the graph is connected or acyclic:

Lemma 10.10.5. *Let G be a nonempty graph, and let v be a vertex of G with $d(v) = 1$. Let $G - v$ be the induced subgraph of G obtained by deleting v and its unique incident edge. Then*

1. *G is connected if and only if $G - v$ is connected.*

2. *G is acyclic if and only if $G - v$ is acyclic.*

Proof. Let w be v's unique neighbor.

If G is connected, for any two vertices s and t, there is a simple s–t path. If neither s nor t is v, this path can't include v, because w would appear

both before and after v in the path, violating simplicity. So for any s, t in $G - v$, there is an s–t path in $G - v$, and $G - v$ is connected.

Conversely, if $G - v$ is connected, then any s and t not equal to v remain connected after adding vw, and if $s = v$, for any t there is a path $w = v_1 \dots v_k = t$, from which we can construct a path $vv_1 \dots v_k = t$ from v to t. The case $t = v$ is symmetric.

If G contains a cycle, then it contains a simple cycle; this cycle can't include v, so $G - v$ also contains the cycle.

Conversely, if $G - v$ contains a cycle, this cycle is also in G. □

Because a graph with two vertices and fewer than one edges is not connected, Lemma 10.10.5 implies that any graph with fewer than $|V| - 1$ edges is not connected.

Corollary 10.10.6. *Let $G = (V, E)$. If $|E| < |V| - 1$, G is not connected.*

Proof. By induction on $n = |V|$.

For the base case, if $n = 0$, then $|E| = 0 \not< n - 1$.

For larger n, suppose that $n \geq 1$ and $|E| < n - 1$. From Lemma 10.10.3 we have $\sum_v d(v) < 2n - 2$, from which it follows that there must be at least one vertex v with $d(v) < 2$. If $d(v) = 0$, then G is not connected. If $d(v) = 1$, then G is connected if and only if $G - v$ is connected. But $G - v$ has $n - 1$ vertices and $|E| - 1 < n - 2$ edges, so by the induction hypothesis, $G - v$ is not connected. So in either case, $|E| < n - 1$ implies G is not connected. □

In the other direction, combining the lemma with the fact that the unique graph K_3 with three vertices and at least three edges is cyclic tells us that any graph with at least as many edges as vertices is cyclic.

Corollary 10.10.7. *Let $G = (V, E)$. If $|E| > |V| - 1$, G contains a cycle.*

Proof. By induction on $n = |V|$.

For $n \leq 2$, $|E| \not> |V - 1|$, so the claim holds vacuously.[3]

For larger n, there are two cases:

1. Some vertex v has degree $d(v) \leq 1$. Let $G' = (V', E') = G - v$. Then $|E'| \geq |E| - 1 > |V| - 2 = |V'| - 1$, and by the induction hypothesis G' contains a cycle. This cycle is also in G.

2. Every vertex v in G has $d(v) \geq 2$. Let's go for a walk: starting at some vertex v_0, choose at each step a vertex v_{i+1} adjacent to v_i that

[3] In fact, no graph with $|V| \leq 2$ contains a cycle, but we don't need to use this.

does not already appear in the walk. This process finishes when we reach a node v_k all of whose neighbors appear in the walk in a previous position. One of these neighbors may be v_{k-1}; but since $d(v_k) \geq 2$, there is another neighbor $v_j \neq v_{k-1}$. So $v_j \dots v_k v_j$ forms a cycle.

□

Now we can prove the full result:

Theorem 10.10.8. *Let $G = (V, E)$ be a nonempty graph. Then any two of the following statements implies the third:*

1. *G is connected.*
2. *G is acyclic.*
3. *$|E| = |V| - 1$.*

Proof. We will use induction on n for some parts of the proof. The base case is when $n = 1$; then all three statements hold always. For larger n, we show:

- (1) and (2) imply (3): Use Corollary 10.10.6 and Corollary 10.10.7.
- (1) and (3) imply (2). From Lemma 10.10.3, $\sum_{v \in V} d(v) = 2(n-1) < 2n$. It follows that there is at least one v with $d(v) \leq 1$. Because G is connected, we must have $d(v) = 1$. So $G' = G - v$ is a graph with $n-2$ edges and $n-1$ vertices. It is connected by Lemma 10.10.5, and thus it is acyclic by the induction hypothesis. Applying the other case of Lemma 10.10.5 in the other direction shows G is also acyclic.
- (2) and (3) imply (1). As in the previous case, G contains a vertex v with $d(v) \leq 1$. If $d(v) = 1$, then $G - v$ is a nonempty graph with $n-2$ edges and $n-1$ vertices that is acyclic by Lemma 10.10.5. It is thus connected by the induction hypothesis, so G is also connected by Lemma 10.10.5. If $d(v) = 0$, then $G - v$ has $n-1$ edges and $n-1$ vertices. From Corollary 10.10.7, $G - v$ contains a cycle, contradicting (2).

□

For an alternative proof based on removing edges, see [Big02, Theorem 15.5]. This also gives the useful fact that removing one edge from a tree gives exactly two components.

10.10.4 Spanning trees

Here's another induction proof on graphs. A **spanning tree** of a nonempty connected graph G is a subgraph of G that includes all vertices and is a tree (i.e., is connected and acyclic).

Theorem 10.10.9. *Every nonempty connected graph has a spanning tree.*

Proof. Let $G = (V, E)$ be a nonempty connected graph. We'll show by induction on $|E|$ that G has a spanning tree. The base case is $|E| = |V| - 1$ (the least value for which G can be connected); then G itself is a tree (by the theorem above). For larger $|E|$, the same theorem gives that G contains a cycle. Let uv be any edge on the cycle, and consider the graph $G - uv$; this graph is connected (since we can route any path that used to go through uv around the other edges of the cycle) and has fewer edges than G, so by the induction hypothesis there is some spanning tree T of $G - uv$. But then T also spans G, so we are done. □

10.10.5 Eulerian cycles

Let's prove the vertex degree characterization of graphs with Eulerian cycles. As in the previous proofs, we'll take the approach of looking for something to pull out of the graph to get a smaller case.

Theorem 10.10.10. *Let G be a connected graph. Then G has an Eulerian cycle if and only if all nodes have even degree.*

Proof.

- (Only if part). Fix some cycle, and orient the edges by the direction that the cycle traverses them. Then in the resulting directed graph we must have $d^-(u) = d^+(u)$ for all u, since every time we enter a vertex we have to leave it again. But then $d(u) = 2d^+(u)$ is even.

- (If part, sketch of proof). Suppose now that $d(u)$ is even for all u. We will construct an Eulerian cycle on all nodes by induction on $|E|$. The base case is when $|E| = 2|V|$ and $G = C_{|V|}$. For a larger graph, choose some starting node u_1, and construct a path $u_1 u_2 \ldots$ by choosing an arbitrary unused edge leaving each u_i; this is always possible for $u_i \neq u_1$ since whenever we reach u_i we have always consumed an even number of edges on previous visits plus one to get to it this time, leaving at least one remaining edge to leave on. Since there are only finitely many edges and we can only use each one once, eventually we must get stuck, and this must occur with $u_k = u_1$ for some k. Now

delete all the edges in $u_1 \dots u_k$ from G, and consider the connected components of $G - (u_1 \dots u_k)$. Removing the cycle reduces $d(v)$ by an even number, so within each such connected component the degree of all vertices is even. It follows from the induction hypothesis that each connected component has an Eulerian cycle. We'll now string these per-component cycles together using our original cycle: while traversing $u_1 \dots, u_k$ when we encounter some component for the first time, we take a detour around the component's cycle. The resulting merged cycle gives an Eulerian cycle for the entire graph.

□

Why doesn't this work for Hamiltonian cycles? The problem is that in a Hamiltonian cycle we have too many choices: out of the $d(u)$ edges incident to u, we will only use two of them. If we pick the wrong two early on, this may prevent us from ever fitting u into a Hamiltonian cycle. So we would need some stronger property of our graph to get Hamiltonicity.