

# TheoryPrep Readings

May 2020

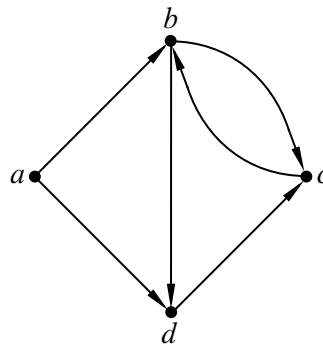
Selected from Mathematics for Computer Science (Lehman, Leighton, Meyer 2017) and Building Blocks for Computer Science (Fleck 2013)

## 10 Directed graphs & Partial Orders

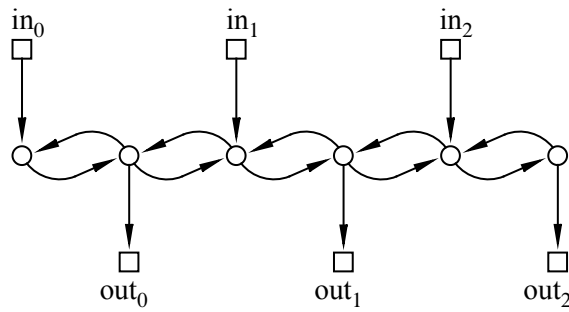
*Directed graphs*, called *digraphs* for short, provide a handy way to represent how things are connected together and how to get from one thing to another by following those connections. They are usually pictured as a bunch of dots or circles with arrows between some of the dots, as in Figure 10.1. The dots are called *nodes* or *vertices* and the lines are called *directed edges* or *arrows*; the digraph in Figure 10.1 has 4 nodes and 6 directed edges.

Digraphs appear everywhere in computer science. For example, the digraph in Figure 10.2 represents a communication net, a topic we’ll explore in depth in Chapter 11. Figure 10.2 has three “in” nodes (pictured as little squares) representing locations where packets may arrive at the net, the three “out” nodes representing destination locations for packets, and the remaining six nodes (pictured with little circles) represent switches. The 16 edges indicate paths that packets can take through the router.

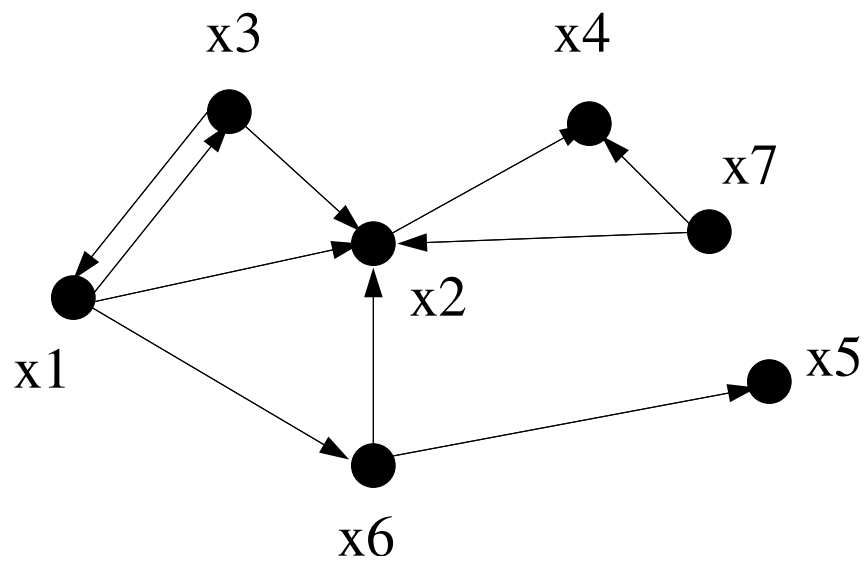
Another place digraphs emerge in computer science is in the hyperlink structure of the World Wide Web. Letting the vertices  $x_1, \dots, x_n$  correspond to web pages, and using arrows to indicate when one page has a hyperlink to another, results in a digraph like the one in Figure 10.3—although the graph of the real World Wide Web would have  $n$  be a number in the billions and probably even the trillions. At first glance, this graph wouldn’t seem to be very interesting. But in 1995, two students at Stanford, Larry Page and Sergey Brin, ultimately became multibillionaires from the realization of how useful the structure of this graph could be in building a search engine. So pay attention to graph theory, and who knows what might happen!



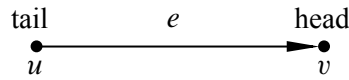
**Figure 10.1** A 4-node directed graph with 6 edges.



**Figure 10.2** A 6-switch packet routing digraph.



**Figure 10.3** Links among Web Pages.



**Figure 10.4** A directed edge  $e = \langle u \rightarrow v \rangle$ . The edge  $e$  starts at the tail vertex  $u$  and ends at the head vertex  $v$ .

**Definition 10.0.1.** A *directed graph*  $G$  consists of a nonempty set  $V(G)$ , called the *vertices* of  $G$ , and a set  $E(G)$ , called the *edges* of  $G$ . An element of  $V(G)$  is called a *vertex*. A vertex is also called a *node*; the words “vertex” and “node” are used interchangeably. An element of  $E(G)$  is called a *directed edge*. A directed edge is also called an “arrow” or simply an “edge.” A directed edge *starts* at some vertex  $u$  called the *tail* of the edge, and *ends* at some vertex  $v$  called the *head* of the edge, as in Figure 10.4. Such an edge can be represented by the ordered pair  $(u, v)$ . The notation  $\langle u \rightarrow v \rangle$  denotes this edge.

There is nothing new in Definition 10.0.1 except for a lot of vocabulary. Formally, a digraph  $G$  is the same as a binary relation on the set,  $V = V(G)$ —that is, a digraph is just a binary relation whose domain and codomain are the same set  $V$ . In fact, we’ve already referred to the arrows in a relation  $G$  as the “graph” of  $G$ . For example, the divisibility relation on the integers in the interval  $[1..12]$  could be pictured by the digraph in Figure 10.5.

## 10.1 Vertex Degrees

The *in-degree* of a vertex in a digraph is the number of arrows coming into it, and similarly its *out-degree* is the number of arrows out of it. More precisely,

**Definition 10.1.1.** If  $G$  is a digraph and  $v \in V(G)$ , then

$$\text{indeg}(v) ::= |\{e \in E(G) \mid \text{head}(e) = v\}|$$

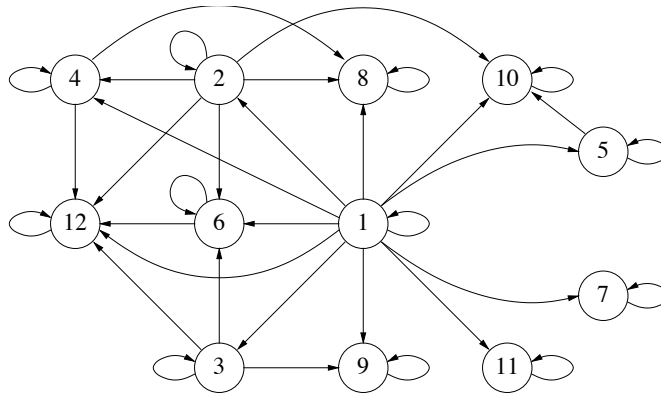
$$\text{outdeg}(v) ::= |\{e \in E(G) \mid \text{tail}(e) = v\}|$$

An immediate consequence of this definition is

**Lemma 10.1.2.**

$$\sum_{v \in V(G)} \text{indeg}(v) = \sum_{v \in V(G)} \text{outdeg}(v).$$

*Proof.* Both sums are equal to  $|E(G)|$ . ■



**Figure 10.5** The Digraph for Divisibility on  $\{1, 2, \dots, 12\}$ .

## 10.2 Walks and Paths

Picturing digraphs with points and arrows makes it natural to talk about following successive edges through the graph. For example, in the digraph of Figure 10.5, you might start at vertex 1, successively follow the edges from vertex 1 to vertex 2, from 2 to 4, from 4 to 12, and then from 12 to 12 twice (or as many times as you like). The sequence of edges followed in this way is called a *walk* through the graph. A *path* is a walk which never visits a vertex more than once. So following edges from 1 to 2 to 4 to 12 is a path, but it stops being a path if you go to 12 again.

The natural way to represent a walk is with the sequence of successive vertices it went through, in this case:

$$1 \ 2 \ 4 \ 12 \ 12 \ 12.$$

However, it is conventional to represent a walk by an alternating sequence of successive vertices and edges, so this walk would formally be

$$1 \ \langle 1 \rightarrow 2 \rangle \ 2 \ \langle 2 \rightarrow 4 \rangle \ 4 \ \langle 4 \rightarrow 12 \rangle \ 12 \ \langle 12 \rightarrow 12 \rangle \ 12 \ \langle 12 \rightarrow 12 \rangle \ 12. \quad (10.1)$$

The redundancy of this definition is enough to make any computer scientist cringe, but it does make it easy to talk about how many times vertices and edges occur on the walk. Here is a formal definition:

**Definition 10.2.1.** A *walk in a digraph* is an alternating sequence of vertices and edges that begins with a vertex, ends with a vertex, and such that for every edge  $\langle u \rightarrow v \rangle$  in the walk, vertex  $u$  is the element just before the edge, and vertex  $v$  is the next element after the edge.

So a walk  $\mathbf{v}$  is a sequence of the form

$$\mathbf{v} ::= v_0 \langle v_0 \rightarrow v_1 \rangle v_1 \langle v_1 \rightarrow v_2 \rangle v_2 \dots \langle v_{k-1} \rightarrow v_k \rangle v_k$$

where  $\langle v_i \rightarrow v_{i+1} \rangle \in E(G)$  for  $i \in [0..k)$ . The walk is said to *start* at  $v_0$ , to *end* at  $v_k$ , and the *length*  $|\mathbf{v}|$  of the walk is defined to be  $k$ .

The walk is a *path* iff all the  $v_i$ 's are different, that is, if  $i \neq j$ , then  $v_i \neq v_j$ .

A *closed walk* is a walk that begins and ends at the same vertex. A *cycle* is a positive length closed walk whose vertices are distinct except for the beginning and end vertices.

Note that a single vertex counts as a length zero path that begins and ends at itself. It also is a closed walk, but does not count as a cycle, since cycles by definition must have positive length. Length one cycles are possible when a node has an arrow leading back to itself. The graph in Figure 10.1 has none, but every vertex in the divisibility relation digraph of Figure 10.5 is in a length one cycle. Length one cycles are sometimes called *self-loops*.

Although a walk is officially an alternating sequence of vertices and edges, it is completely determined just by the sequence of successive vertices on it, or by the sequence of edges on it. We will describe walks in these ways whenever it's convenient. For example, for the graph in Figure 10.1,

- $(a, b, d)$ , or simply  $abd$ , is a (vertex-sequence description of a) length two path,
- $(\langle a \rightarrow b \rangle, \langle b \rightarrow d \rangle)$ , or simply  $\langle a \rightarrow b \rangle \langle b \rightarrow d \rangle$ , is (an edge-sequence description of) the same length two path,
- $abcbcd$  is a length four walk,
- $dcbcbcd$  is a length five closed walk,
- $bdc b$  is a length three cycle,
- $\langle b \rightarrow c \rangle \langle c \rightarrow b \rangle$  is a length two cycle, and
- $\langle c \rightarrow b \rangle \langle b \leftarrow a \rangle \langle a \rightarrow d \rangle$  is *not* a walk. A walk is not allowed to follow edges in the wrong direction.

If you walk for a while, stop for a rest at some vertex, and then continue walking, you have broken a walk into two parts. For example, stopping to rest after following two edges in the walk (10.1) through the divisibility graph breaks the walk into the first part of the walk

$$1 \langle 1 \rightarrow 2 \rangle 2 \langle 2 \rightarrow 4 \rangle 4 \tag{10.2}$$

from 1 to 4, and the rest of the walk

$$4 \langle 4 \rightarrow 12 \rangle 12 \langle 12 \rightarrow 12 \rangle 12 \langle 12 \rightarrow 12 \rangle 12. \quad (10.3)$$

from 4 to 12, and we’ll say the whole walk (10.1) is the *mergewalks* (10.2) and (10.3). In general, if a walk  $\mathbf{f}$  ends with a vertex  $v$  and a walk  $\mathbf{r}$  starts with the same vertex  $v$  we’ll say that their *merge*  $\mathbf{f} \hat{\vee} \mathbf{r}$  is the walk that starts with  $\mathbf{f}$  and continues with  $\mathbf{r}$ .<sup>1</sup> Two walks can only be merged if the first walk ends at the same vertex  $v$  with which the second one walk starts. Sometimes it’s useful to name the node  $v$  where the walks merge; we’ll use the notation  $\mathbf{f} \hat{\vee}_v \mathbf{r}$  to describe the merge of a walk  $\mathbf{f}$  that ends at  $v$  with a walk  $\mathbf{r}$  that begins at  $v$ .

A consequence of this definition is that

**Lemma 10.2.2.**

$$|\mathbf{f} \hat{\vee} \mathbf{r}| = |\mathbf{f}| + |\mathbf{r}|.$$

In the next section we’ll get mileage out of walking this way.

### 10.2.1 Finding a Path

If you were trying to walk somewhere quickly, you’d know you were in trouble if you came to the same place twice. This is actually a basic theorem of graph theory.

**Theorem 10.2.3.** *The shortest walk from one vertex to another is a path.*

*Proof.* If there is a walk from vertex  $u$  to another vertex  $v \neq u$ , then by the Well Ordering Principle, there must be a minimum length walk  $\mathbf{w}$  from  $u$  to  $v$ . We claim  $\mathbf{w}$  is a path.

To prove the claim, suppose to the contrary that  $\mathbf{w}$  is not a path, meaning that some vertex  $x$  occurs twice on this walk. That is,

$$\mathbf{w} = \mathbf{e} \hat{\vee}_x \mathbf{f} \hat{\vee}_x \mathbf{g}$$

for some walks  $\mathbf{e}, \mathbf{f}, \mathbf{g}$  where the length of  $\mathbf{f}$  is positive. But then “deleting”  $\mathbf{f}$  yields a strictly shorter walk

$$\mathbf{e} \hat{\vee}_x \mathbf{g}$$

from  $u$  to  $v$ , contradicting the minimality of  $\mathbf{w}$ . ■

**Definition 10.2.4.** The *distance*,  $\text{dist}(u, v)$ , in a graph from vertex  $u$  to vertex  $v$  is the length of a shortest path from  $u$  to  $v$ .

<sup>1</sup>It’s tempting to say the *merge* is the concatenation of the two walks, but that wouldn’t quite be right because if the walks were concatenated, the vertex  $v$  would appear twice in a row where the walks meet.

As would be expected, this definition of distance satisfies:

**Lemma 10.2.5.** [*The Triangle Inequality*]

$$\text{dist}(u, v) \leq \text{dist}(u, x) + \text{dist}(x, v)$$

for all vertices  $u, v, x$  with equality holding iff  $x$  is on a shortest path from  $u$  to  $v$ .

Of course, you might expect this property to be true, but distance has a technical definition and its properties can’t be taken for granted. For example, unlike ordinary distance in space, the distance from  $u$  to  $v$  is typically different from the distance from  $v$  to  $u$ . So, let’s prove the Triangle Inequality

*Proof.* To prove the inequality, suppose  $\mathbf{f}$  is a shortest path from  $u$  to  $x$  and  $\mathbf{r}$  is a shortest path from  $x$  to  $v$ . Then by Lemma 10.2.2,  $\mathbf{f} \hat{\ } \mathbf{r}$  is a walk of length  $\text{dist}(u, x) + \text{dist}(x, v)$  from  $u$  to  $v$ , so this sum is an upper bound on the length of the shortest path from  $u$  to  $v$  by Theorem 10.2.3.

Proof of the “iff” is in Problem 10.3. ■

Finally, the relationship between walks and paths extends to closed walks and cycles:

**Lemma 10.2.6.** *The shortest positive length closed walk through a vertex is a cycle through that vertex.*

The proof of Lemma 10.2.6 is essentially the same as for Theorem 10.2.3; see Problem 10.4.

---

## 10.3 Adjacency Matrices

If a graph  $G$  has  $n$  vertices  $v_0, v_1, \dots, v_{n-1}$ , a useful way to represent it is with an  $n \times n$  matrix of zeroes and ones called its *adjacency matrix*  $A_G$ . The  $ij$ th entry of the adjacency matrix,  $(A_G)_{ij}$ , is 1 if there is an edge from vertex  $v_i$  to vertex  $v_j$  and 0 otherwise. That is,

$$(A_G)_{ij} ::= \begin{cases} 1 & \text{if } \langle v_i \rightarrow v_j \rangle \in E(G), \\ 0 & \text{otherwise.} \end{cases}$$



For example, let  $H$  be the 4-node graph shown in Figure 10.1. Its adjacency matrix  $A_H$  is the  $4 \times 4$  matrix:

$$A_H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 1 \\ b & 0 & 0 & 1 & 1 \\ c & 0 & 1 & 0 & 0 \\ d & 0 & 0 & 1 & 0 \end{array}$$

A payoff of this representation is that we can use matrix powers to count numbers of walks between vertices. For example, there are two length two walks between vertices  $a$  and  $c$  in the graph  $H$ :

$$\begin{array}{l} a \langle a \rightarrow b \rangle b \langle b \rightarrow c \rangle c \\ a \langle a \rightarrow d \rangle d \langle d \rightarrow c \rangle c \end{array}$$

**We covered this in the video  
if you want to skip to  
the next green arrow**



and these are the only length two walks from  $a$  to  $c$ . Also, there is exactly one length two walk from  $b$  to  $c$  and exactly one length two walk from  $c$  to  $c$  and from  $d$  to  $b$ , and these are the only length two walks in  $H$ . It turns out we could have read these counts from the entries in the matrix  $(A_H)^2$ :

$$(A_H)^2 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 0 & 2 & 1 \\ b & 0 & 1 & 1 & 0 \\ c & 0 & 0 & 1 & 1 \\ d & 0 & 1 & 0 & 0 \end{array}$$

More generally, the matrix  $(A_G)^k$  provides a count of the number of length  $k$  walks between vertices in any digraph  $G$  as we'll now explain.

**Definition 10.3.1.** The length- $k$  walk counting matrix for an  $n$ -vertex graph  $G$  is the  $n \times n$  matrix  $C$  such that

$$C_{uv} ::= \text{the number of length-}k \text{ walks from } u \text{ to } v. \quad (10.4)$$

Notice that the adjacency matrix  $A_G$  is the length-1 walk counting matrix for  $G$ , and that  $(A_G)^0$ , which by convention is the identity matrix, is the length-0 walk counting matrix.

**Theorem 10.3.2.** If  $C$  is the length- $k$  walk counting matrix for a graph  $G$ , and  $D$  is the length- $m$  walk counting matrix, then  $CD$  is the length  $k + m$  walk counting matrix for  $G$ .

According to this theorem, the square  $(A_G)^2$  of the adjacency matrix is the length two walk counting matrix for  $G$ . Applying the theorem again to  $(A_G)^2 A_G$  shows that the length-3 walk counting matrix is  $(A_G)^3$ . More generally, it follows by induction that

**Corollary 10.3.3.** *The length- $k$  counting matrix of a digraph  $G$  is  $(A_G)^k$ , for all  $k \in \mathbb{N}$ .*

In other words, you can determine the number of length  $k$  walks between any pair of vertices simply by computing the  $k$ th power of the adjacency matrix!

That may seem amazing, but the proof uncovers this simple relationship between matrix multiplication and numbers of walks.

*Proof of Theorem 10.3.2.* Any length  $(k+m)$  walk between vertices  $u$  and  $v$  begins with a length  $k$  walk starting at  $u$  and ending at some vertex  $w$  followed by a length  $m$  walk starting at  $w$  and ending at  $v$ . So the number of length  $(k+m)$  walks from  $u$  to  $v$  that go through  $w$  at the  $k$ th step equals the number  $C_{uw}$  of length  $k$  walks from  $u$  to  $w$ , times the number  $D_{wv}$  of length  $m$  walks from  $w$  to  $v$ . We can get the total number of length  $(k+m)$  walks from  $u$  to  $v$  by summing, over all possible vertices  $w$ , the number of such walks that go through  $w$  at the  $k$ th step. In other words,

$$\# \text{length } (k+m) \text{ walks from } u \text{ to } v = \sum_{w \in V(G)} C_{uw} \cdot D_{wv} \quad (10.5)$$

But the right-hand side of (10.5) is precisely the definition of  $(CD)_{uv}$ . Thus,  $CD$  is indeed the length- $(k+m)$  walk counting matrix. ■

Start

### 10.3.1 Shortest Paths

The relation between powers of the adjacency matrix and numbers of walks is cool—to us math nerds at least—but a much more important problem is finding shortest paths between pairs of nodes. For example, when you drive home for vacation, you generally want to take the shortest-time route.

One simple way to find the lengths of all the shortest paths in an  $n$ -vertex graph  $G$  is to compute the successive powers of  $A_G$  one by one up to the  $n-1$ st, watching for the first power at which each entry becomes positive. That’s because Theorem 10.3.2 implies that the length of the shortest path, if any, between  $u$  and  $v$ , that is, the distance from  $u$  to  $v$ , will be the smallest value  $k$  for which  $(A_G)^k_{uv}$  is nonzero, and if there is a shortest path, its length will be  $\leq n-1$ . Refinements of this idea lead to methods that find shortest paths in reasonably efficient ways. The methods apply as well to weighted graphs, where edges are labelled with weights or costs and the objective is to find least weight, cheapest paths. These refinements

are typically covered in introductory algorithm courses, and we won’t go into them any further.

## 10.4 Walk Relations

A basic question about a digraph is whether there is a way to get from one particular vertex to another. So for any digraph  $G$  we are interested in a binary relation  $G^*$ , called the *walk relation* on  $V(G)$ , where

$$u G^* v ::= \text{there is a walk in } G \text{ from } u \text{ to } v. \quad (10.6)$$

Similarly, there is a *positive walk relation*

$$u G^+ v ::= \text{there is a positive length walk in } G \text{ from } u \text{ to } v. \quad (10.7)$$

**Definition 10.4.1.** When there is a walk from vertex  $v$  to vertex  $w$ , we say that  $w$  is *reachable* from  $v$ , or equivalently, that  $v$  is *connected* to  $w$ .

### 10.4.1 Composition of Relations

There is a simple way to extend composition of functions to composition of relations, and this gives another way to talk about walks and paths in digraphs.

**Definition 10.4.2.** Let  $R : B \rightarrow C$  and  $S : A \rightarrow B$  be binary relations. Then the composition of  $R$  with  $S$  is the binary relation  $(R \circ S) : A \rightarrow C$  defined by the rule


$$a (R \circ S) c ::= \exists b \in B. (a S b) \text{ AND } (b R c). \quad (10.8)$$

This agrees with the Definition 4.3.1 of composition in the special case when  $R$  and  $S$  are functions.<sup>2</sup>

Remembering that a digraph is a binary relation on its vertices, it makes sense to compose a digraph  $G$  with itself. Then if we let  $G^n$  denote the composition of  $G$  with itself  $n$  times, it’s easy to check (see Problem 10.11) that  $G^n$  is the *length- $n$  walk relation*:

$$a G^n b \quad \text{iff} \quad \text{there is a length } n \text{ walk in } G \text{ from } a \text{ to } b.$$

<sup>2</sup>The reversal of the order of  $R$  and  $S$  in (10.8) is not a typo. This is so that relational composition generalizes function composition. The value of function  $f$  composed with function  $g$  at an argument  $x$  is  $f(g(x))$ . So in the composition  $f \circ g$ , the function  $g$  is applied first.


 This section has a lot of formalism. It’s less important, so read only if you have time

This even works for  $n = 0$ , with the usual convention that  $G^0$  is the *identity relation*  $\text{Id}_{V(G)}$  on the set of vertices.<sup>3</sup> Since there is a walk iff there is a path, and every path is of length at most  $|V(G)| - 1$ , we now have<sup>4</sup>

$$G^* = G^0 \cup G^1 \cup G^2 \cup \dots \cup G^{|V(G)|-1} = (G \cup G^0)^{|V(G)|-1}. \quad (10.9)$$

The final equality points to the use of repeated squaring as a way to compute  $G^*$  with  $\log n$  rather than  $n - 1$  compositions of relations.

## 10.5 Directed Acyclic Graphs & Scheduling

 You can skim this section. It's okay if you don't understand everything. The most important thing is to understand the definition of a DAG and topological sort

Some of the prerequisites of MIT computer science subjects are shown in Figure 10.6. An edge going from subject  $s$  to subject  $t$  indicates that  $s$  is listed in the catalogue as a direct prerequisite of  $t$ . Of course, before you can take subject  $t$ , you have to take not only subject  $s$ , but also all the prerequisites of  $s$ , and any prerequisites of those prerequisites, and so on. We can state this precisely in terms of the positive walk relation: if  $D$  is the direct prerequisite relation on subjects, then subject  $u$  has to be completed before taking subject  $v$  iff  $u D^+ v$ .

Of course it would take forever to graduate if this direct prerequisite graph had a positive length closed walk. We need to forbid such closed walks, which by Lemma 10.2.6 is the same as forbidding cycles. So, the direct prerequisite graph among subjects had better be *acyclic*:

**Definition 10.5.1.** A *directed acyclic graph (DAG)* is a directed graph with no cycles.

DAGs have particular importance in computer science. They capture key concepts used in analyzing task scheduling and concurrency control. When distributing a program across multiple processors, we're in trouble if one part of the program needs an output that another part hasn't generated yet! So let's examine DAGs and their connection to scheduling in more depth.

<sup>3</sup>The *identity relation*  $\text{Id}_A$  on a set  $A$  is the equality relation:

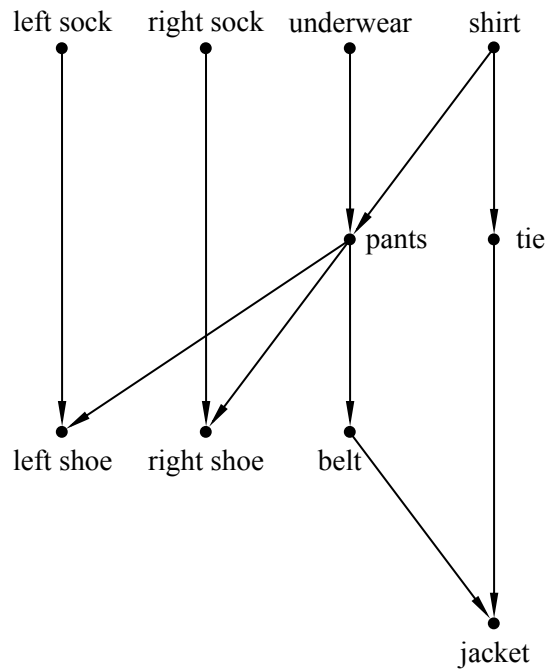
$$a \text{Id}_A b \quad \text{iff} \quad a = b,$$

for  $a, b \in A$ .

<sup>4</sup>Equation (10.9) involves a harmless abuse of notation: we should have written

$$\text{graph}(G^*) = \text{graph}(G^0) \cup \text{graph}(G^1) \dots$$





**Figure 10.7** DAG describing which clothing items have to be put on before others.

### 10.5.1 Scheduling

In a scheduling problem, there is a set of tasks, along with a set of constraints specifying that starting certain tasks depends on other tasks being completed beforehand. We can map these sets to a digraph, with the tasks as the nodes and the direct prerequisite constraints as the edges.

For example, the DAG in Figure 10.7 describes how a man might get dressed for a formal occasion. As we describe above, vertices correspond to garments and the edges specify which garments have to be put on before which others.

When faced with a set of prerequisites like this one, the most basic task is finding an order in which to perform all the tasks, one at a time, while respecting the dependency constraints. Ordering tasks in this way is known as *topological sorting*.

**Definition 10.5.2.** A *topological sort* of a finite DAG is a list of all the vertices such that each vertex  $v$  appears earlier in the list than every other vertex reachable from  $v$ .

There are many ways to get dressed one item at a time while obeying the constraints of Figure 10.7. We have listed two such topological sorts in Figure 10.8. In

underwear	left sock
shirt	shirt
pants	tie
belt	underwear
tie	right sock
jacket	pants
left sock	right shoe
right sock	belt
left shoe	jacket
right shoe	left shoe
(a)	(b)

**Figure 10.8** Two possible topological sorts of the prerequisites described in Figure 10.7

fact, we can prove that *every* finite DAG has a topological sort. You can think of this as a mathematical proof that you can indeed get dressed in the morning.

Topological sorts for finite DAGs are easy to construct by starting from *minimal* elements:

**Definition 10.5.3.** An vertex  $v$  of a DAG  $D$  is *minimum* iff every other vertex is reachable from  $v$ .

A vertex  $v$  is *indexminimal vertex!directed graph—textbf minimal* iff  $v$  is not reachable from any other vertex.

It can seem peculiar to use the words “minimum” and “minimal” to talk about vertices that start paths. These words come from the perspective that a vertex is “smaller” than any other vertex it connects to. We’ll explore this way of thinking about DAGs in the next section, but for now we’ll use these terms because they are conventional.

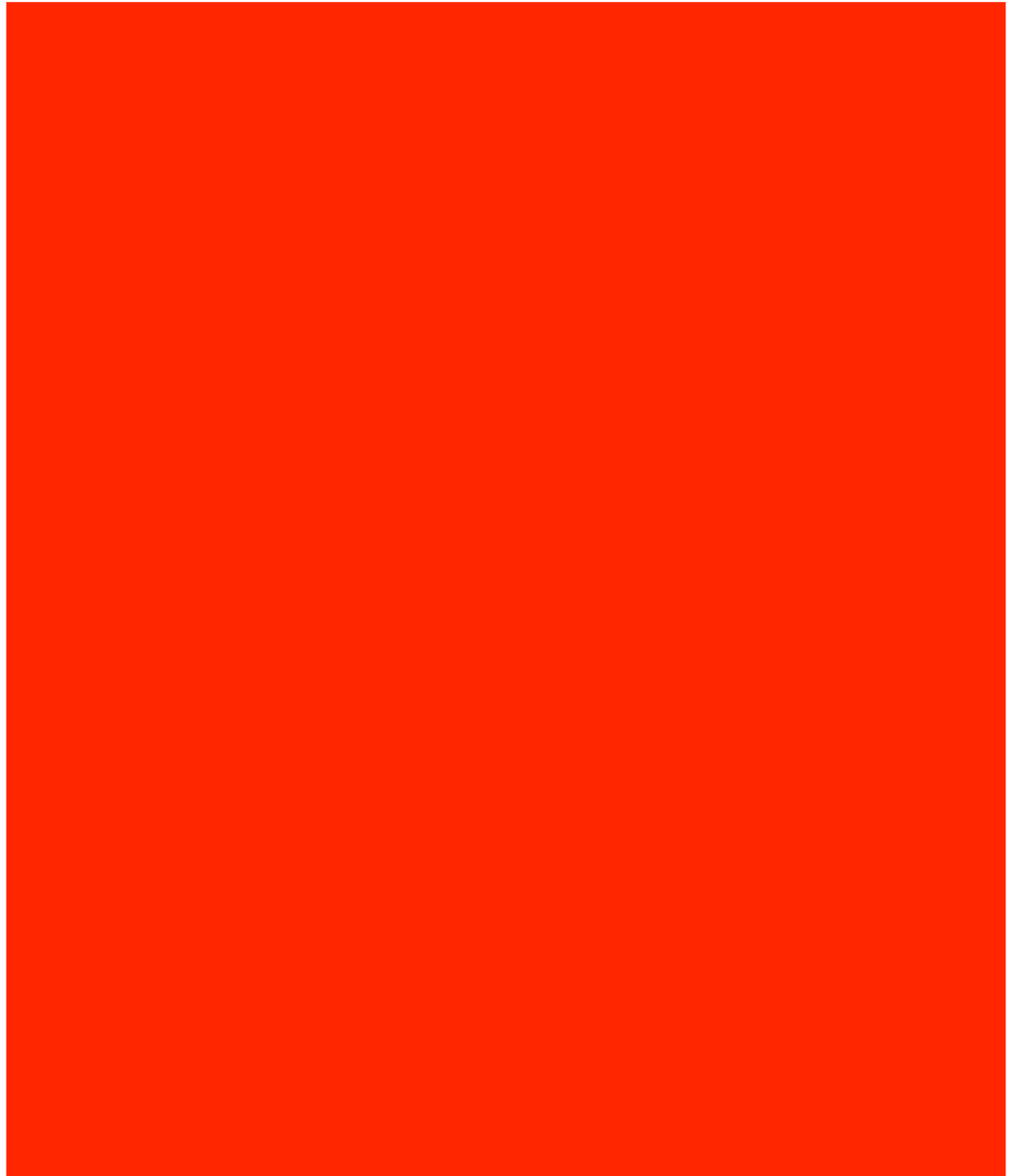
One peculiarity of this terminology is that a DAG may have no minimum element but lots of minimal elements. In particular, the clothing example has four minimal elements: leftsock, rightsock, underwear, and shirt.

To build an order for getting dressed, we pick one of these minimal elements—say, shirt. Now there is a new set of minimal elements; the three elements we didn’t chose as step 1 are still minimal, and once we have removed shirt, tie becomes minimal as well. We pick another minimal element, continuing in this way until all elements have been picked. The sequence of elements in the order they were picked will be a topological sort. This is how the topological sorts above were constructed.

So our construction shows:

**Theorem 10.5.4.** *Every finite DAG has a topological sort.*

There are many other ways of constructing topological sorts. For example, instead of starting from the minimal elements at the beginning of paths, we could build a topological sort starting from *maximal* elements at the end of paths. In fact, we could build a topological sort by picking vertices arbitrarily from a finite DAG and simply inserting them into the list wherever they will fit.<sup>5</sup>

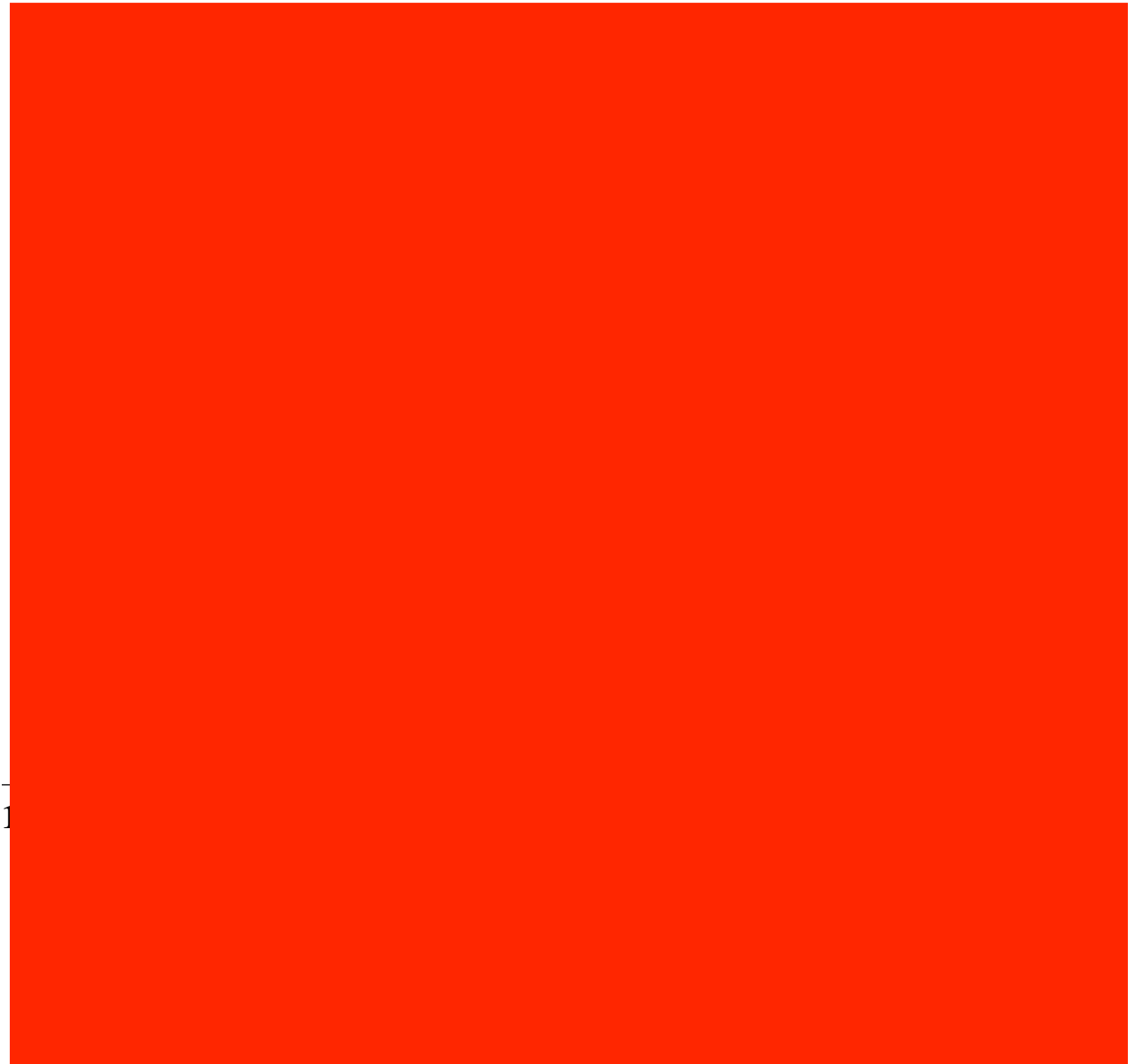




---

## 12 Simple Graphs

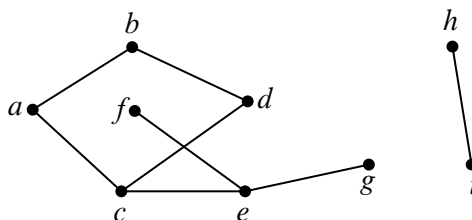
*Simple graphs* model relationships that are *symmetric*, meaning that the relationship is mutual. Examples of such mutual relationships are being married, speaking the same language, not speaking the same language, occurring during overlapping time intervals, or being connected by a conducting wire. They come up in all sorts of applications, including scheduling, constraint satisfaction, computer graphics, and communications, but we'll start with an application designed to get your attention:



ends at vertex  $w$ , a simple graph only has an undirected edge  $\langle v-w \rangle$  that connects  $v$  and  $w$ .

**Definition 12.1.1.** A simple graph  $G$  consists of a nonempty set,  $V(G)$ , called the *vertices* of  $G$ , and a set  $E(G)$  called the *edges*. An element of  $V(G)$  is called a *vertex*. An element of  $E(G)$  is an *undirected edge* or simply an “edge.” An undirected edge has two vertices  $u \neq v$  called its *endpoints*. Such an edge can be represented by the two element set  $\{u, v\}$ . The notation  $\langle u-v \rangle$  denotes this edge.

Both  $\langle u-v \rangle$  and  $\langle v-u \rangle$  describe the same undirected edge, whose endpoints are  $u$  and  $v$ .



**Figure 12.1** An example of a graph with 9 vertices and 8 edges.

For example, let  $H$  be the graph pictured in Figure 12.1. The vertices of  $H$  correspond to the nine dots in Figure 12.1, that is,

$$V(H) = \{a, b, c, d, e, f, g, h, i\}.$$

The edges correspond to the eight lines, that is,

$$E(H) = \{ \langle a-b \rangle, \langle a-c \rangle, \langle b-d \rangle, \langle c-d \rangle, \langle c-e \rangle, \langle e-f \rangle, \langle e-g \rangle, \langle h-i \rangle \}.$$

Mathematically, that’s all there is to the graph  $H$ .

**Definition 12.1.2.** Two vertices in a simple graph are said to be *adjacent* iff they are the endpoints of the same edge, and an edge is said to be *incident* to each of its endpoints. The number of edges incident to a vertex  $v$  is called the *degree* of the vertex and is denoted by  $\deg(v)$ . Equivalently, the degree of a vertex is the number of vertices adjacent to it.

For example, for the graph  $H$  of Figure 12.1, vertex  $a$  is adjacent to vertex  $b$ , and  $b$  is adjacent to  $d$ . The edge  $\langle a-c \rangle$  is incident to its endpoints  $a$  and  $c$ . Vertex  $h$  has degree 1,  $d$  has degree 2, and  $\deg(e) = 3$ . It is possible for a vertex to have degree 0, in which case it is not adjacent to any other vertices. A simple graph  $G$  does not need to have any edges at all.  $|E(G)|$  could be zero, implying that the

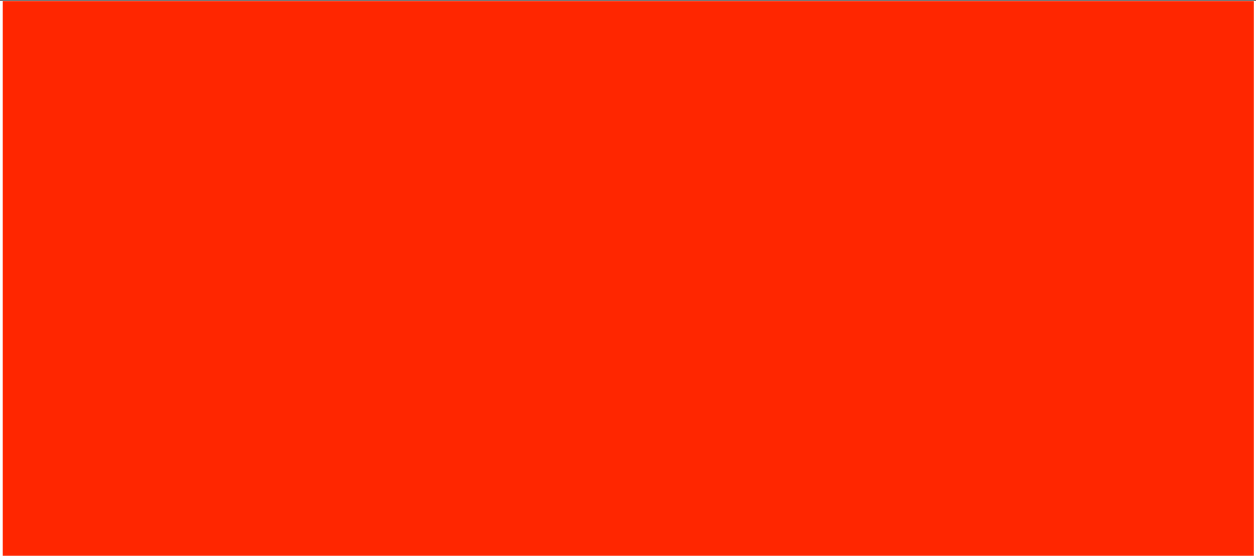
degree of every vertex would also be zero. But a simple graph must have at least one vertex— $|V(G)|$  is required to be at least one.

Note that in a simple graph there cannot be more than one edge between the same two vertices.<sup>1</sup> Also *self-loops*—edges that begin and at the same vertex—are not allowed in simple graphs.

*For the rest of this chapter we’ll use “graphs” as an abbreviation for “simple graphs.”*

A synonym for “vertices” is “*nodes*,” and we’ll use these words interchangeably. Simple graphs are sometimes called *networks*, edges are sometimes called *arcs* or *lines*. We mention this as a “heads up” in case you look at other graph theory literature; we won’t use these words.





### 12.2.1 Handshaking Lemma

The previous argument hinged on the connection between a sum of degrees and the number of edges. There is a simple connection between these in any graph:

**Lemma 12.2.1.** *The sum of the degrees of the vertices in a graph equals twice the number of edges.*

*Proof.* Every edge contributes two to the sum of the degrees, one for each of its endpoints. ■

We refer to Lemma 12.2.1 as the *Handshaking Lemma*: if we total up the number of people each person at a party shakes hands with, the total will be twice the number of handshakes that occurred.

---

## 12.3 Some Common Graphs

Some graphs come up so frequently that they have names. A *complete graph*  $K_n$  has  $n$  vertices and an edge between every two vertices, for a total of  $n(n-1)/2$  edges. For example,  $K_5$  is shown in Figure 12.3.

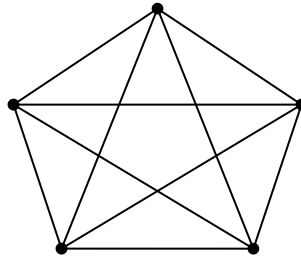
The *empty graph* has no edges at all. For example, the five-vertex empty graph is shown in Figure 12.4.

An  $n$ -vertex graph containing  $n-1$  edges in sequence is known as a *line graph*  $L_n$ . More formally,  $L_n$  has

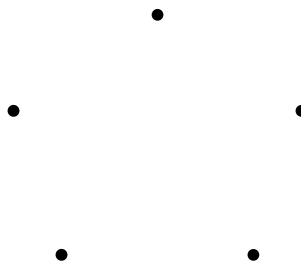
$$V(L_n) = \{v_1, v_2, \dots, v_n\}$$

and

$$E(L_n) = \{ \langle v_1-v_2 \rangle, \langle v_2-v_3 \rangle, \dots, \langle v_{n-1}-v_n \rangle \}$$



**Figure 12.3**  $K_5$ : the complete graph on five nodes.



**Figure 12.4** An empty graph with five nodes.

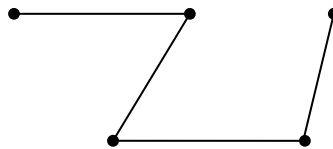
For example,  $L_5$  is pictured in Figure 12.5.

There is also a one-way infinite line graph  $L_\infty$  which can be defined by letting the nonnegative integers  $\mathbb{N}$  be the vertices with edges  $\langle k, (k+1) \rangle$  for all  $k \in \mathbb{N}$ .

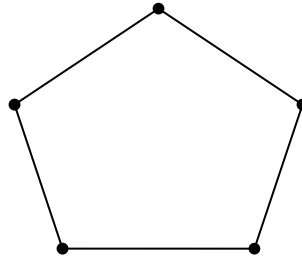
If we add the edge  $\langle v_n, v_1 \rangle$  to the line graph  $L_n$ , we get a graph called a *length- $n$  cycle*  $C_n$ . Figure 12.6 shows a picture of length-5 cycle.

## 12.4 Isomorphism

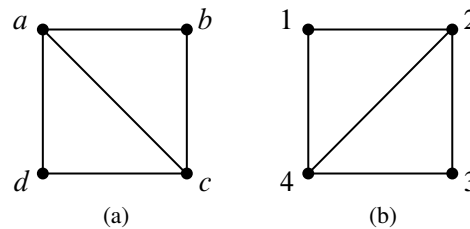
Two graphs that look different might actually be the same in a formal sense. For example, the two graphs in Figure 12.7 are both four-vertex, five-edge graphs and



**Figure 12.5**  $L_5$ : a 5-vertex line graph.



**Figure 12.6**  $C_5$ : a 5-node cycle graph.



**Figure 12.7** Two Isomorphic graphs.

you get graph (b) by a  $90^\circ$  clockwise rotation of graph (a).

Strictly speaking, these graphs are different mathematical objects, but this difference doesn’t reflect the fact that the two graphs can be described by the same picture—except for the labels on the vertices. This idea of having the same picture “up to relabeling” can be captured neatly by adapting Definition 10.7.1 of isomorphism of digraphs to handle simple graphs. An isomorphism between two graphs is an edge-preserving bijection between their sets of vertices:

**Definition 12.4.1.** An isomorphism between graphs  $G$  and  $H$  is a bijection  $f : V(G) \rightarrow V(H)$  such that

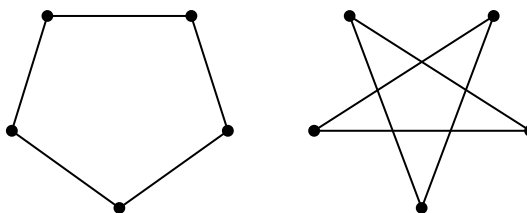
$$\langle u-v \rangle \in E(G) \text{ IFF } \langle f(u)-f(v) \rangle \in E(H)$$

for all  $u, v \in V(G)$ . Two graphs are isomorphic when there is an isomorphism between them.

Here is an isomorphism  $f$  between the two graphs in Figure 12.7:

$$\begin{array}{ll} f(a) ::= 2 & f(b) ::= 3 \\ f(c) ::= 4 & f(d) ::= 1. \end{array}$$

You can check that there is an edge between two vertices in the graph on the left if and only if there is an edge between the two corresponding vertices in the graph on the right.



**Figure 12.8** Isomorphic  $C_5$  graphs.

Two isomorphic graphs may be drawn very differently. For example, Figure 12.8 shows two different ways of drawing  $C_5$ .

Notice that if  $f$  is an isomorphism between  $G$  and  $H$ , then  $f^{-1}$  is an isomorphism between  $H$  and  $G$ . Isomorphism is also transitive because the composition of isomorphisms is an isomorphism. In fact, isomorphism is an equivalence relation.

Isomorphism preserves the connection properties of a graph, abstracting out what the vertices are called, what they are made out of, or where they appear in a drawing of the graph. More precisely, a property of a graph is said to be *preserved under isomorphism* if whenever  $G$  has that property, every graph isomorphic to  $G$  also has that property. For example, since an isomorphism is a bijection between sets of vertices, isomorphic graphs must have the same number of vertices. What’s more, if  $f$  is a graph isomorphism that maps a vertex  $v$  of one graph to the vertex  $f(v)$  of an isomorphic graph, then by definition of isomorphism, every vertex adjacent to  $v$  in the first graph will be mapped by  $f$  to a vertex adjacent to  $f(v)$  in the isomorphic graph. Thus,  $v$  and  $f(v)$  will have the same degree. If one graph has a vertex of degree four and another does not, then they can’t be isomorphic. In fact, they can’t be isomorphic if the number of degree-four vertices in each of the graphs is not the same.

Looking for preserved properties can make it easy to determine that two graphs are not isomorphic, or to guide the search for an isomorphism when there is one. It’s generally easy in practice to decide whether two graphs are isomorphic. However, no one has yet found a procedure for determining whether two graphs are isomorphic that is *guaranteed* to run in polynomial time on all pairs of graphs.<sup>3</sup>

Having such a procedure would be useful. For example, it would make it easy to search for a particular molecule in a database given the molecular bonds. On the other hand, knowing there is no such efficient procedure would also be valuable: secure protocols for encryption and remote authentication can be built on the

<sup>3</sup>A procedure runs in *polynomial time* when it needs an amount of time of at most  $p(n)$ , where  $n$  is the total number of vertices and  $p()$  is a fixed polynomial.

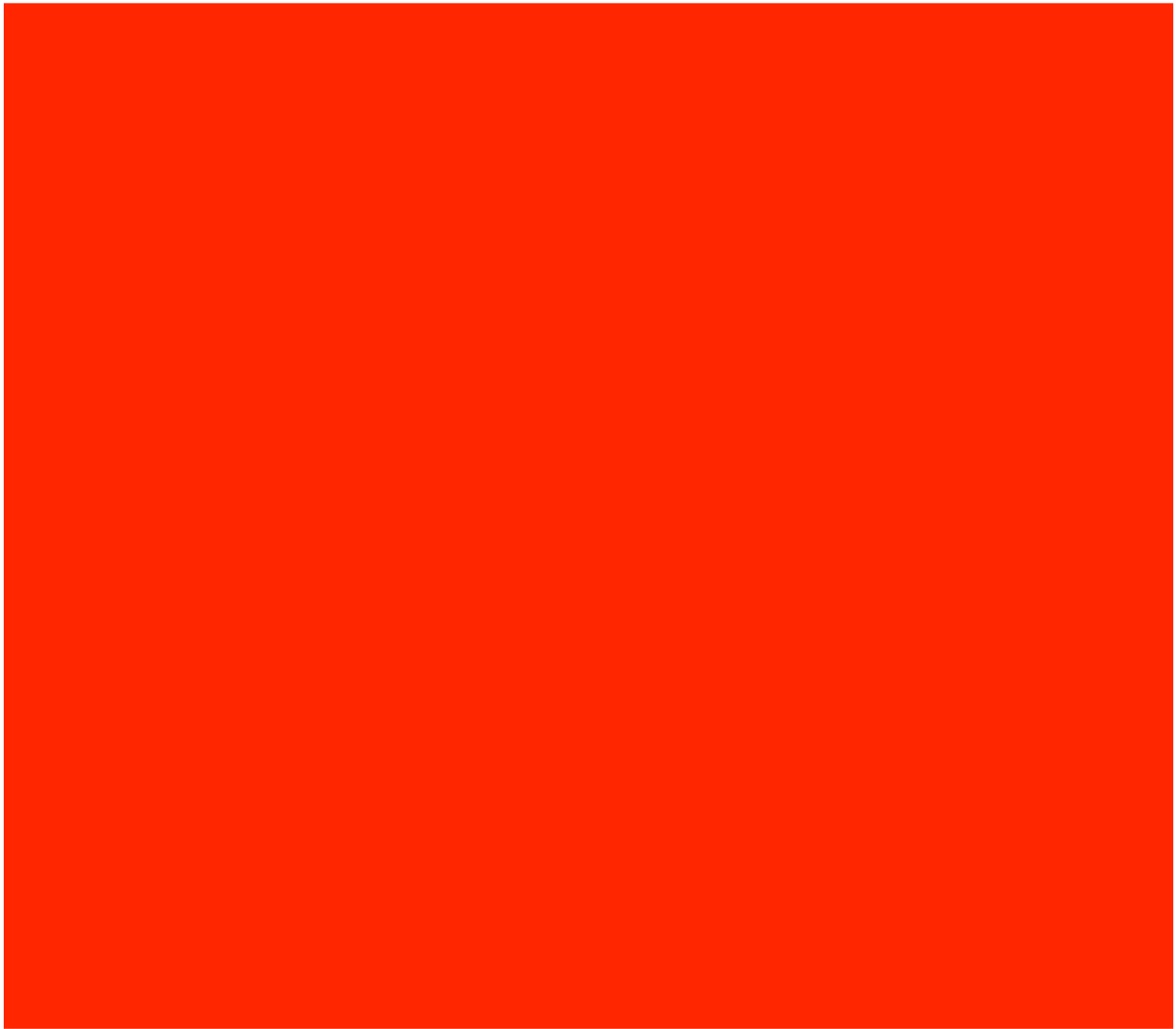
hypothesis that graph isomorphism is computationally exhausting.

The definitions of bijection and isomorphism apply to infinite graphs as well as finite graphs, as do most of the results in the rest of this chapter. But graph theory focuses mostly on finite graphs, and we will too.

*In the rest of this chapter we'll assume graphs are finite.*

We've actually been taking isomorphism for granted ever since we wrote “ $K_n$  has  $n$  vertices...” at the beginning of Section [12.3](#).

*Graph theory is all about properties preserved by isomorphism.*





## 12.7 Walks in Simple Graphs

### 12.7.1 Walks, Paths, Cycles

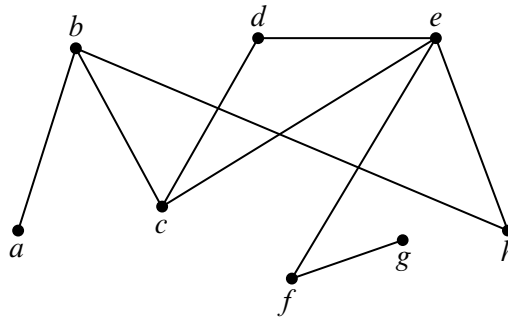
Walks and paths in simple graphs are essentially the same as in digraphs. We just modify the digraph definitions using undirected edges instead of directed ones. For example, the formal definition of a walk in a simple graph is virtually the same as the Definition 10.2.1 of a walk in a digraph:

**Definition 12.7.1.** A *walk in a simple graph  $G$*  is an alternating sequence of vertices and edges that begins with a vertex, ends with a vertex, and such that for every edge  $\langle u—v \rangle$  in the walk, one of the endpoints  $u, v$  is the element just before the edge, and the other endpoint is the next element after the edge. The *length of a walk* is the total number of occurrences of edges in it.

So a walk  $\mathbf{v}$  is a sequence of the form

$$\mathbf{v} ::= v_0 \langle v_0—v_1 \rangle v_1 \langle v_1—v_2 \rangle v_2 \dots \langle v_{k-1}—v_k \rangle v_k$$

where  $\langle v_i—v_{i+1} \rangle \in E(G)$  for  $i \in [0..k)$ . The walk is said to *start* at  $v_0$ , to *end*



**Figure 12.14** A graph with 3 cycles:  $bhecb$ ,  $cdec$ ,  $bcdehb$ .

at  $v_k$ , and the *length*,  $|\mathbf{v}|$ , of the walk is  $k$ . The walk is a *path* iff all the  $v_i$ 's are different, that is, if  $i \neq j$ , then  $v_i \neq v_j$ .

A walk that begins and ends at the same vertex is a *closed walk*. A single vertex counts as a length zero closed walk as well as a length zero path.

A *cycle* can be represented by a closed walk of length three or more whose vertices are distinct except for the beginning and end vertices.

Note that in contrast to digraphs, we don't count length two closed walks as cycles in simple graphs. That's because a walk going back and forth on the same edge is neither interesting nor important. There are also no closed walks of length one, since simple graphs don't have self-loops.

As in digraphs, the length of a walk is the number of occurrences of *edges* in the walk, which is *one less* than the number of occurrences of vertices. For example, the graph in Figure 12.14 has a length 6 path through the seven successive vertices  $abcdefg$ . This is the longest path in the graph. The graph in Figure 12.14 also has three cycles through successive vertices  $bhecb$ ,  $cdec$  and  $bcdehb$ .

This is cool but not integral

### 12.7.2 Cycles as Subgraphs

We don't intend that cycles have a beginning or an end, so *any* of the paths that go around a cycle can represent it. For example, in the graph in Figure 12.14, the cycle starting at  $b$  and going through vertices  $bcdehb$  can also be described as starting at  $d$  and going through  $dehbcd$ . Furthermore, cycles in simple graphs don't have a direction:  $dcbhed$  describes the same cycle as though it started and ended at  $d$  but went in the opposite direction.

A precise way to explain which closed walks represent the same cycle is to define a cycle to be *subgraph* isomorphic to a cycle graph  $C_n$ .

**Definition 12.7.2.** A graph  $G$  is said to be a *subgraph* of a graph  $H$  if  $V(G) \subseteq V(H)$  and  $E(G) \subseteq E(H)$ .

For example, the one-edge graph  $G$  where

$$V(G) = \{g, h, i\} \quad \text{and} \quad E(G) = \{ \langle h-i \rangle \}$$

is a subgraph of the graph  $H$  in Figure 12.1. On the other hand, any graph containing an edge  $\langle g-h \rangle$  will not be a subgraph of  $H$  because this edge is not in  $E(H)$ . Another example is an empty graph on  $n$  nodes, which will be a subgraph of an  $L_n$  with the same set of vertices; similarly,  $L_n$  is a subgraph of  $C_n$ , and  $C_n$  is a subgraph of  $K_n$ .

**Definition 12.7.3.** For  $n \geq 3$ , let  $C_n$  be the graph with vertices  $[0..n)$  and edges

$$\langle 0-1 \rangle, \langle 1-2 \rangle, \dots, \langle (n-2)-n-1 \rangle, \langle n-1-0 \rangle.$$

A *cycle* of a graph  $G$  is a subgraph of  $G$  that is isomorphic to  $C_n$  for some  $n \geq 3$ .

This definition formally captures the idea that cycles don’t have direction or beginnings or ends.

## 12.8 Connectivity

Read this def

**Definition 12.8.1.** Two vertices are *connected* in a graph when there is a path that begins at one and ends at the other. By convention, every vertex is connected to itself by a path of length zero. A *graph is connected* when every pair of vertices are connected.

