## 11.7 Coloring

In Section 11.2, we used edges to indicate an affinity between a pair of nodes. But there are lots of situations in which edges will correspond to *conflicts* between nodes. Exam scheduling is a typical example.

### 11.7.1 An Exam Scheduling Problem

Each term, the MIT Schedules Office must assign a time slot for each final exam. This is not easy, because some students are taking several classes with finals, and (even at MIT) a student can take only one test during a particular time slot. The Schedules Office wants to avoid all conflicts. Of course, you can make such a schedule by having every exam in a different slot, but then you would need hundreds of slots for the hundreds of courses, and the exam period would run all year! So, the Schedules Office would also like to keep exam period short.

The Schedules Office's problem is easy to describe as a graph. There will be a vertex for each course with a final exam, and two vertices will be adjacent exactly when some student is taking both courses. For example, suppose we need to schedule exams for 6.041, 6.042, 6.002, 6.003 and 6.170. The scheduling graph might appear as in Figure 11.12.

6.002 and 6.042 cannot have an exam at the same time since there are students in both courses, so there is an edge between their nodes. On the other hand, 6.042 and 6.170 can have an exam at the same time if they're taught at the same time (which they sometimes are), since no student can be enrolled in both (that is, no student *should* be enrolled in both when they have a timing conflict).

We next identify each time slot with a color. For example, Monday morning

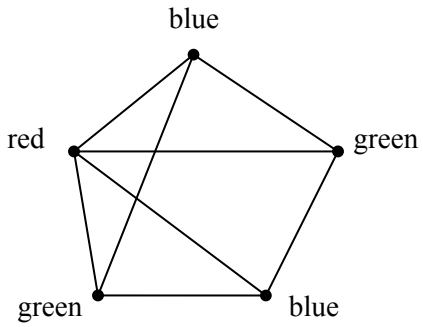


**Figure 11.13** A 3-coloring of the exam graph from Figure 11.12.

is red, Monday afternoon is blue, Tuesday morning is green, etc. Assigning an exam to a time slot is then equivalent to coloring the corresponding vertex. The main constraint is that *adjacent vertices must get different colors*—otherwise, some student has two exams at the same time. Furthermore, in order to keep the exam period short, we should try to color all the vertices using as *few different colors as possible*. As shown in Figure 11.13, three colors suffice for our example.

The coloring in Figure 11.13 corresponds to giving one final on Monday morning (red), two Monday afternoon (blue), and two Tuesday morning (green). Can we use fewer than three colors? No! We can't use only two colors since there is a triangle in the graph, and three vertices in a triangle must all have different colors.

This is an example of a *graph coloring* problem: given a graph $G$, assign colors to each node such that adjacent nodes have different colors. A color assignment with this property is called a *valid coloring* of the graph—a "*coloring*," for short. A graph $G$ is *$k$-colorable* if it has a coloring that uses at most $k$ colors.

**Definition 11.7.1.** The minimum value of $k$ for which a graph, $G$, has a valid coloring is called its *chromatic number*, $\chi(G)$.

So $G$ is $k$-colorable iff $\chi(G) \leq k$.

In general, trying to figure out if you can color a graph with a fixed number of colors can take a long time. It's a classic example of a problem for which no fast algorithms are known. In fact, it is easy to check if a coloring works, but it seems really hard to find it. (If you figure out how, then you can get a $1 million Clay prize.)

### 11.7.2 Some Coloring Bounds

There are some simple properties of graphs that give useful bounds on colorability.

The simplest property is being a cycle: an even-length closed cycle is 2-colorable.

Cycles in simple graphs by convention have positive length and so are not 1-colorable. So

$$\chi(C_{\text{even}}) = 2.$$

On the other hand, an odd-length cycle requires 3 colors, that is,

$$\chi(C_{\text{odd}}) = 3. \tag{11.3}$$

You should take a moment to think about why this equality holds.

Another simple example is a complete graph $K_n$:

$$\chi(K_n) = n$$

since no two vertices can have the same color.

Being bipartite is another property closely related to colorability. If a graph is bipartite, then you can color it with 2 colors using one color for the nodes on the "left" and a second color for the nodes on the "right." Conversely, graphs with chromatic number 2 are all bipartite with all the vertices of one color on the "left" and those with the other color on the right. Since only graphs with no edges—the *empty graphs*—have chromatic number 1, we have:

**Lemma 11.7.2.** *A graph, $G$, with at least one edge is bipartite iff $\chi(G) = 2$.*

The chromatic number of a graph can also be shown to be small if the vertex degrees of the graph are small. In particular, if we have an upper bound on the degrees of all the vertices in a graph, then we can easily find a coloring with only one more color than the degree bound.

**Theorem 11.7.3.** *A graph with maximum degree at most $k$ is $(k+1)$-colorable.*

Since $k$ is the only nonnegative integer valued variable mentioned in the theorem, you might be tempted to try to prove this theorem using induction on $k$. Unfortunately, this approach leads to disaster—we don't know of any reasonable way to do this and expect it would ruin your week if you tried it on a problem set. When you encounter such a disaster using induction on graphs, it is usually best to change what you are inducting on. In graphs, typical good choices for the induction parameter are $n$, the number of nodes, or $e$, the number of edges.

*Proof of Theorem 11.7.3.* We use induction on the number of vertices in the graph, which we denote by $n$. Let $P(n)$ be the proposition that an $n$-vertex graph with maximum degree at most $k$ is $(k+1)$-colorable.

**Base case** ($n = 1$): A 1-vertex graph has maximum degree 0 and is 1-colorable, so $P(1)$ is true.

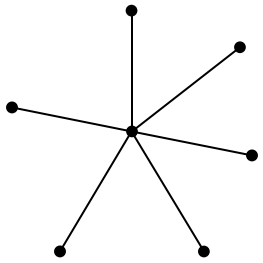

**Figure 11.14** A 7-node star graph.

**Inductive step**: Now assume that $P(n)$ is true, and let $G$ be an $(n+1)$-vertex graph with maximum degree at most $k$. Remove a vertex $v$ (and all edges incident to it), leaving an $n$-vertex subgraph, $H$. The maximum degree of $H$ is at most $k$, and so $H$ is $(k+1)$-colorable by our assumption $P(n)$. Now add back vertex $v$. We can assign $v$ a color (from the set of $k+1$ colors) that is different from all its adjacent vertices, since there are at most $k$ vertices adjacent to $v$ and so at least one of the $k+1$ colors is still available. Therefore, $G$ is $(k+1)$-colorable. This completes the inductive step, and the theorem follows by induction. ■

Sometimes $k+1$ colors is the best you can do. For example, $\chi(K_n) = n$ and every node in $K_n$ has degree $k = n-1$ and so this is an example where Theorem 11.7.3 gives the best possible bound. By a similar argument, we can show that Theorem 11.7.3 gives the best possible bound for *any* graph with degree bounded by $k$ that has $K_{k+1}$ as a subgraph.

But sometimes $k+1$ colors is far from the best that you can do. For example, the $n$-node *star graph* shown in Figure 11.14 has maximum degree $n-1$ but can be colored using just 2 colors.

### 11.7.3 Why coloring?

One reason coloring problems frequently arise in practice is because scheduling conflicts are so common. For example, at Akamai, a new version of software is deployed over each of 65,000 servers every few days. The updates cannot be done at the same time since the servers need to be taken down in order to deploy the software. Also, the servers cannot be handled one at a time, since it would take forever to update them all (each one takes about an hour). Moreover, certain pairs of servers cannot be taken down at the same time since they have common critical functions. This problem was eventually solved by making a 65,000-node conflict graph and coloring it with 8 colors—so only 8 waves of install are needed!

Another example comes from the need to assign frequencies to radio stations. If

two stations have an overlap in their broadcast area, they can't be given the same frequency. Frequencies are precious and expensive, so you want to minimize the number handed out. This amounts to finding the minimum coloring for a graph whose vertices are the stations and whose edges connect stations with overlapping areas.

Coloring also comes up in allocating registers for program variables. While a variable is in use, its value needs to be saved in a register. Registers can be reused for different variables but two variables need different registers if they are referenced during overlapping intervals of program execution. So register allocation is the coloring problem for a graph whose vertices are the variables: vertices are adjacent if their intervals overlap, and the colors are registers. Once again, the goal is to minimize the number of colors needed to color the graph.

Finally, there's the famous map coloring problem stated in Proposition 1.1.6. The question is how many colors are needed to color a map so that adjacent territories get different colors? This is the same as the number of colors needed to color a graph that can be drawn in the plane without edges crossing. A proof that four colors are enough for *planar* graphs was acclaimed when it was discovered about thirty years ago. Implicit in that proof was a 4-coloring procedure that takes time proportional to the number of vertices in the graph (countries in the map).

Surprisingly, it's another of those million dollar prize questions to find an efficient procedure to tell if a planar graph really *needs* four colors, or if three will actually do the job. A proof that testing 3-colorability of graphs is as hard as the million dollar SAT problem is given in Problem 11.39; this turns out to be true even for planar graphs. (It is easy to tell if a graph is 2-colorable, as explained in Section 11.9.2.) In Chapter 12, we'll develop enough planar graph theory to present an easy proof that all planar graphs are 5-colorable.

## 11.8 Simple Walks

### 11.8.1 Walks, Paths, Cycles in Simple Graphs

Walks and paths in simple graphs are esentially 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 a virtually the same as the Definition 9.2.1 of a walk in a digraph:

**Definition 11.8.1.** 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

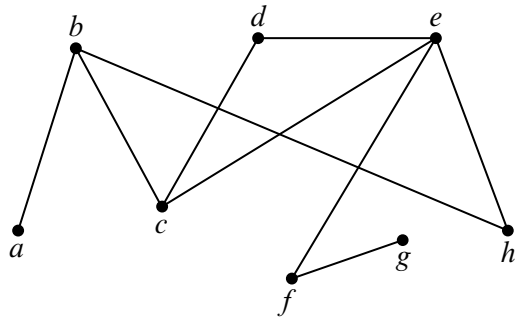


**Figure 11.15** *A graph with 3 cycles: bhecb, cdec, bcdehb.*

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, **v**, is a sequence of the form

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

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

A *cycle* is 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 always possible in a simple graph, and it has no importance. Also, there are no closed walks of length one, since simple graphs don't have self loops.

As in digraphs, the length of a walk is *one less* than the number of occurrences of vertices in it. For example, the graph in Figure 11.15 has a length 6 path through the seven successive vertices $abcdefg$. This is the longest path in the graph. The graph in Figure 11.15 also has three cycles through successive vertices $bhecb$, $cdec$, and $bcdehb$.

### 11.8.2 Cycles as Subgraphs

A cycle does not really have a beginning or an end, so it can be described by *any* of the paths that go around it. For example, in the graph in Figure 11.15, 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 describe the same cycle is to define cycle as a subgraph instead of as a closed walk. Specifically, we could define a cycle in $G$ to be a *subgraph* of $G$ that looks like a length-$n$ cycle for $n \geq 3$.

**Definition 11.8.2.** 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

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

is a subgraph of the graph $H$ in Figure 11.1. On the other hand, any graph containing an edge $\langle g\text{—}h\rangle$ 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 nodes; similarly, $L_n$ is a subgraph of $C_n$, and $C_n$ is a subgraph of $K_n$.

**Definition 11.8.3.** For $n \geq 3$, let $C_n$ be the graph with vertices $1, \ldots, n$ and edges

$$\langle 1\text{—}2\rangle, \ \langle 2\text{—}3\rangle, \ \ldots, \ \langle (n-1)\text{—}n\rangle, \ \langle n\text{—}1\rangle.$$

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.

## 11.9 Connectivity

**Definition 11.9.1.** *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.

### 11.9.1 Connected Components

Being connected is usually a good property for a graph to have. For example, it could mean that it is possible to get from any node to any other node, or that it is possible to communicate between any pair of nodes, depending on the application.

But not all graphs are connected. For example, the graph where nodes represent cities and edges represent highways might be connected for North American cities, but would surely not be connected if you also included cities in Australia. The same is true for communication networks like the internet—in order to be protected from viruses that spread on the internet, some government networks are completely isolated from the internet.

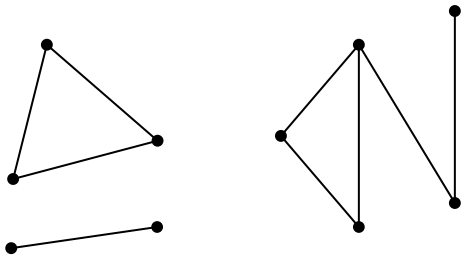

**Figure 11.16** One graph with 3 connected components.

Another example is shown in Figure 11.16, which looks like a picture of three graphs, but is intended to be a picture of *one* graph. This graph consists of three pieces (subgraphs). Each piece by itself is connected, but there are no paths between vertices in different pieces. These connected pieces of a graph are called its *connected components*.

**Definition 11.9.2.** A *connected component* of a graph is a subgraph consisting of some vertex and every node and edge that is connected to that vertex.

So, a graph is connected iff it has exactly one connected component. At the other extreme, the empty graph on $n$ vertices has $n$ connected components.

### 11.9.2 Odd Cycles and 2-Colorability

We have already seen that determining the chromatic number of a graph is a challenging problem. There is one special case where this problem is very easy, namely, when the graph is 2-colorable.

**Theorem 11.9.3.** *The following graph properties are equivalent:*

1. *The graph contains an odd length cycle.*
2. *The graph is not 2-colorable.*
3. *The graph contains an odd length closed walk.*

In other words, if a graph has any one of the three properties above, then it has all of the properties.

We will show the following implications among these properties:

1. IMPLIES 2. IMPLIES 3. IMPLIES 1.

So each of these properties implies the other two, which means they all are equivalent.

**1 IMPLIES 2** *Proof.* This follows from equation 11.3. ■

**2 IMPLIES 3** If we prove this implication for connected graphs, then it will hold for an arbitrary graph because it will hold for each connected component. So we can assume that $G$ is connected.

*Proof.* Pick an arbitrary vertex $r$ of $G$. Since $G$ is connected, for every node $u \in V(G)$, there will be a walk $\mathbf{w}_u$ starting at $u$ and ending at $r$. Assign colors to vertices of $G$ as follows:

$$\text{color}(u) = \begin{cases} \text{black}, & \text{if } |\mathbf{w}_u| \text{ is even,} \\ \text{white}, & \text{otherwise.} \end{cases}$$

Now since $G$ is not colorable, this can't be a valid coloring. So there must be an edge between two nodes $u$ and $v$ with the same color. But in that case

$$\mathbf{w}_u \widehat{\ } \text{reverse}(\mathbf{w}_v) \widehat{\ } \langle v\text{—}u \rangle$$

is a closed walk starting and ending at $u$, and its length is

$$|\mathbf{w}_u| + |\mathbf{w}_v| + 1$$

which is odd. ■

**3 IMPLIES 1** *Proof.* Since there is an odd length closed walk, the WOP implies there is an odd length closed walk $\mathbf{w}$ of minimum length. We claim $\mathbf{w}$ must be a cycle. To show this, assume to the contrary that $\mathbf{w}$ is not a cycle, so there is a repeat vertex occurrence besides the start and end. There are then two cases to consider depending on whether the additional repeat is different from, or the same as, the start vertex.

In the first case, the start vertex has an extra occurrence. That is,

$$\mathbf{w} = \mathbf{f}\, \widehat{x}\, \mathbf{r}$$

for some positive length walks **f** and **r** that begin and end at $x$. Since

$$|\mathbf{w}| = |\mathbf{f}| + |\mathbf{r}|$$

is odd, exactly one of **f** and **r** must have odd length, and that one will be an odd length closed walk shorter than **w**, a contradiction.

In the second case,

$$\mathbf{w} = \mathbf{f}\,\widehat{y}\,\mathbf{g}\,\widehat{y}\,\mathbf{r}$$

where **f** is a walk from $x$ to $y$ for some $y \neq x$, and **r** is a walk from $y$ to $x$, and $|\mathbf{g}| > 0$. Now **g** cannot have odd length or it would be an odd-length closed walk shorter than **w**. So **g** has even length. That implies that $\mathbf{f}\,\widehat{y}\,\mathbf{r}$ must be an odd-length closed walk shorter than **w**, again a contradiction.

This completes the proof of Theorem 11.9.3. ■

Theorem 11.9.3 turns out to be useful, since bipartite graphs come up fairly often in practice. We'll see examples when we talk about planar graphs in Chapter 12.

### 11.9.3 *k*-connected Graphs

If we think of a graph as modeling cables in a telephone network, or oil pipelines, or electrical power lines, then we not only want connectivity, but we want connectivity that survives component failure. So more generally, we want to define how strongly two vertices are connected. One measure of connection strength is how many links must fail before connectedness fails. In particular, two vertices are *k-edge connected* when it takes at least $k$ "edge-failures" to disconnect them. More precisely:

**Definition 11.9.4.** Two vertices in a graph are *k-edge connected* when they remain connected in every subgraph obtained by deleting up to $k-1$ edges. A graph is $k$-edge connected when it has more than one vertex, and pair of distinct vertices in the graph are $k$- connected.

Notice that according to Definition 11.9.4, if a graph is $k$-connected, it is also $j$-connected for $j \leq k$. This convenient convention implies that two vertices are connected according to definition 11.9.1 iff they are 1-edge connected according to Definition 11.9.4. From now on we'll drop the "edge" modifier and just say "$k$-connected."[9]

[9]There is a corresponding definition of $k$-vertex connectedness based on deleting vertices rather than edges. Graph theory texts usually use "$k$-connected" as shorthand for "$k$-vertex connected." But edge-connectedness will be enough for us.

For example, in the graph in figure 11.15, vertices $c$ and $e$ are 3-connected, $b$ and $e$ are 2-connected, $g$ and $e$ are 1 connected, and no vertices are 4-connected. The graph as a whole is only 1-connected. A complete graph, $k_n$, is $(n-1)$-connected. Every cycle is 2-connected.

The idea of a *cut edge* is a useful way to explain 2-connectivity.

**Definition 11.9.5.** If two vertices are connected in a graph $G$, but not connected when an edge $e$ is removed, then $e$ is called a *cut edge* of $G$.

So a graph with more than one vertex is 2-connected iff it is connected and has no cut edges. The following Lemma is another immediate consequence of the definition:

**Lemma 11.9.6.** *An edge is a cut edge iff it is not on a cycle.*

More generally, if two vertices are connected by $k$ edge-disjoint paths—that is, no edge occurs in two paths—then they must be $k$-connected, since at least one edge will have to be removed from each of the paths before they could disconnect. A fundamental fact, whose ingenious proof we omit, is Menger's theorem which confirms that the converse is also true: if two vertices are $k$-connected, then there are $k$ edge-disjoint paths connecting them. It takes some ingenuity to prove this just for the case $k = 2$.

### 11.9.4 The Minimum Number of Edges in a Connected Graph

The following theorem says that a graph with few edges must have many connected components.

**Theorem 11.9.7.** *Every graph, $G$, has at least $|V(G)| - |E(G)|$ connected components.*

Of course for Theorem 11.9.7 to be of any use, there must be fewer edges than vertices.

*Proof.* We use induction on the number, $k$, of edges. Let $P(k)$ be the proposition that

> every graph, $G$, with $k$ edges has at least $|V(G)| - k$ connected components.

**Base case** ($k = 0$): In a graph with 0 edges, each vertex is itself a connected component, and so there are exactly $|V(G)| = |V(G)| - 0$ connected components. So $P(0)$ holds.

**Inductive step**:

Let $G_e$ be the graph that results from removing an edge, $e \in E(G)$. So $G_e$ has $k$ edges, and by the induction hypothesis $P(k)$, we may assume that $G_e$ has at least $|V(G)| - k$-connected components. Now add back the edge $e$ to obtain the original graph $G$. If the endpoints of $e$ were in the same connected component of $G_e$, then $G$ has the same sets of connected vertices as $G_e$, so $G$ has at least $|V(G)| - k > |V(G)| - (k+1)$ components. Alternatively, if the endpoints of $e$ were in different connected components of $G_e$, then these two components are merged into one component in $G$, while all other components remain unchanged, so that $G$ has one fewer connected component than $G_e$. That is, $G$ has at least $(|V(G)| - k) - 1 = |V(G)| - (k+1)$ connected components. So in either case, $G$ has at least $|V(G)| - (k+1)$ components, as claimed.

This completes the inductive step and hence the entire proof by induction. ■

**Corollary 11.9.8.** *Every connected graph with $n$ vertices has at least $n-1$ edges.*

A couple of points about the proof of Theorem 11.9.7 are worth noticing. First, we used induction on the number of edges in the graph. This is very common in proofs involving graphs, as is induction on the number of vertices. When you're presented with a graph problem, these two approaches should be among the first you consider.

The second point is more subtle. Notice that in the inductive step, we took an arbitrary $(k+1)$-edge graph, threw out an edge so that we could apply the induction assumption, and then put the edge back. You'll see this shrink-down, grow-back process very often in the inductive steps of proofs related to graphs. This might seem like needless effort: why not start with an $k$-edge graph and add one more to get an $(k+1)$-edge graph? That would work fine in this case, but opens the door to a nasty logical error called *buildup error*, illustrated in Problem 11.48.

6.042J / 18.062J Mathematics for Computer Science
Spring 2015