Chapter 16

Introduction to Graph Theory

In mathematical english the word *graph* has two meanings. In one context, a graph is a visual device that describes the relationship between two or more variables, as in the graph of a function; these are the kinds of graphs encountered in algebra and calculus. In its other meaning, a graph is a system of interconnecting nodes. In this second context, a graph can be visualized as set of nodes, and lines connecting pairs of nodes. Graphs are useful because they model relationships between entities in ways that cannot be expressed by arithmetic, algebra and calculus alone. A molecule is a graph: the nodes are atoms and the edges are bonds between atoms. A map of an airline's flights is a graph: the nodes are cities and the edges are routes between cities. Your family tree is a graph. In computer science, data structures of all types can be thought of and analyzed as graphs.

The mathematical study of properties of graphs is called *Graph Theory.* Graph theory is relatively new as a mathematical discipline. Although a few papers from the later half of the 1800's dealt with what we now call graphs, it wasn't until around 1950 that graph theory began to emerge as an independent field of study. Originally it had a somewhat recreational flavor, often dealing with mathematical questions motivated by games and puzzles. But it is a perfect language for expressing structures in computer science, and the field grew with the computer revolution.

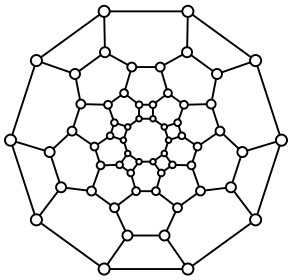


Fig. 16.1 A graph.

Today graph theory is a vast and ever-expanding field of study. This chapter, then, can only be a brief introduction. We will lay out the main definitions, prove a few theorems, and examine some graph algorithms. You will build on this platform if you continue with discrete mathematics beyond this text.

This chapter also presents an opportunity to practice the various proof techniques that you learned in earlier chapters.

16.1 Graphs and Subgraphs

Technically, a node in a graph is called a **vertex**. The plural of vertex is **vertices**. Graphs are usually denoted by upper-case letters, like G or H. To specify a graph G, we need only to describe its vertices and edges. Here is the exact definition.

Definition 16.1. A **graph** G is a finite set $V(G)$ of objects, called **vertices**, together with a set $E(G)$ of two-element subsets of $V(G)$, called **edges**. An edge $\{x, y\} \in E(G)$ is abbreviated as xy or yx.

For example, consider the graph G with $V(G) = \{a, b, c, d, e\}$ and $E(G) = \{ab, bc, cd, de, ea, ad, eb\}$. We can make a picture of G by drawing a dot or small circle for each vertex in $V(G)$, and then drawing line segments joining any two vertices that appear as an edge in $E(G)$. Figure 16.2 shows three different pictures of this graph. In drawing a graph we do not mind if the edges cross each other, but we never allow an edge touch any vertex that is not one of its endpoints.

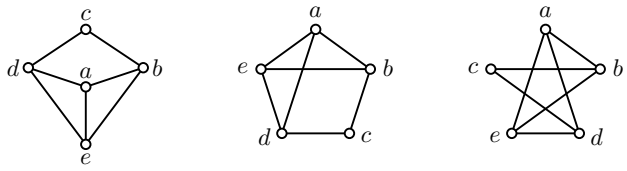
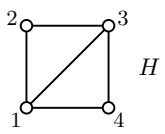


Fig. 16.2 Three drawings of the same graph G.

We say two vertices $x, y \in V(G)$ are **adjacent** if $xy \in E(G)$. Thus for the graph G drawn in Figure 16.2, a and b are adjacent, but a and c are not. The **endpoints** an edge xy are the vertices x and y. Two edges are **incident** if they share an endpoint. So ab and bc are incident, but ab and cd are not. A vertex and an edge are **incident** if the vertex is an endpoint of the edge. Thus the edge ab is incident with both of the vertices a and b.

Please note that the sets $V(G)$ and $E(G)$ contain no information other than what the vertices are, and which pairs of them are adjacent. There are endless ways to draw the same graph.

Often we specify a graph simply by drawing it. The figure below conveys that H is a graph with vertex set $V(H) = \{1, 2, 3, 4\}$ and $E(G) = \{12, 23, 34, 41, 13\}$. Drawing the picture is easier than writing out $V(G)$ and $E(G)$.



Some graphs so common that we reserve special symbols for them. Given a positive integer n, the **path** P_n is the graph with n vertices $v_1, v_2, \ldots, v_n$ and edges $E(P_n) = \{v_1v_2,\ v_2v_3,\ v_3v_4, \ldots v_{n-1}v_n\}$. Thus P_n has n vertices and $n-1$ edges. The **cycle** C_n is the graph with n vertices $v_1, v_2, \ldots, v_n$ and edges $E(C_n) = \{v_1v_2,\ v_2v_3,\ v_3v_4,\ \ldots\ v_{n-1}v_n,\ v_nv_1\}$. See Figure 16.3. We say a path or cycle is **even** if it has an even number of vertices; otherwise it is **odd**.

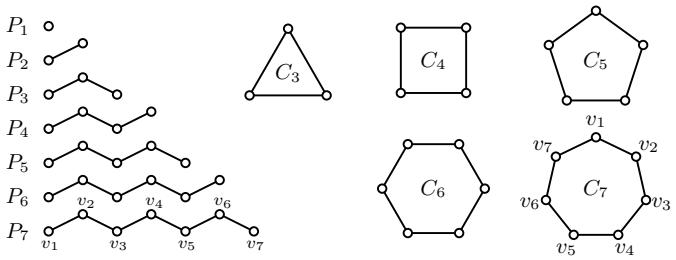


Fig. 16.3 Paths and cycles

The **complete graph** K_n is the graph with n vertices and an edge joining every pair of vertices, as in Figure 16.4. The number of edges in K_n is $\binom{n}{2} = \frac{n(n-1)}{2}$, the number of ways to choose two endpoints from n vertices.

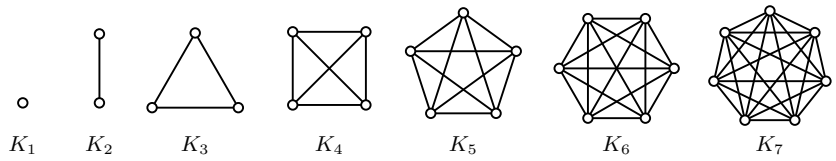


Fig. 16.4 Complete graphs

For two positive integers m and n, the **complete bipartite graph** $K_{m,n}$ is the graph whose vertex set $V(K_{m,n}) = X \cup Y$ is the union of two disjoint sets X and Y of sizes m and n, respectively, and with $E(K_{m,n}) = \{xy : x \in X,\ y \in Y\}$.

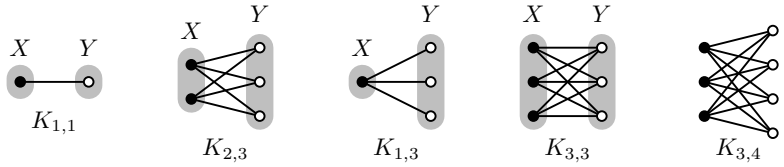
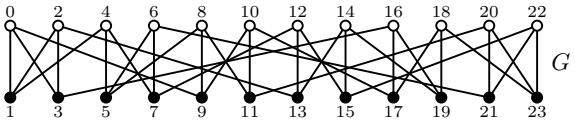


Fig. 16.5 Complete bipartite graphs.

The complete bipartite graphs $K_{m,n}$ are special graphs in a larger class of graphs called *bipartite* graphs. A graph G is said to be **bipartite** if it is possible to find some partition $V(G) = X \cup Y$ of $V(G)$ into two disjoint sets, so that each edge of G has one endpoint in X and the other in Y. For example, here is a bipartite graph G, with $X = \{0, 2, 4, \ldots, 22\}$ and $Y = \{1, 3, 5, \ldots, 23\}$:



Think of a bipartite graph as one whose vertices can be colored black (X) and white (Y) so that each edge joins a black vertex to a white one. The graph by no means has to be drawn with the black and white vertices lined up on different sides. Here is a picture of the exact same graph G above.

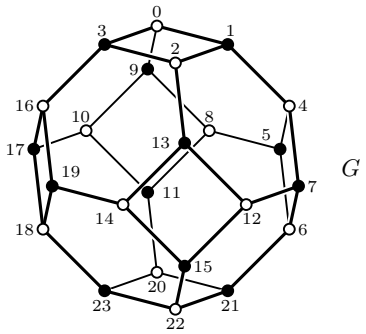


Figure 16.6 indicates that the even cycles $C_4, C_6, C_8, \ldots$ are bipartite. But the odd cycles $C_3, C_5, C_7, \ldots$ are *not* bipartite. Alternating black and white around the cycle forces two adjacent vertices of the same color at the end.

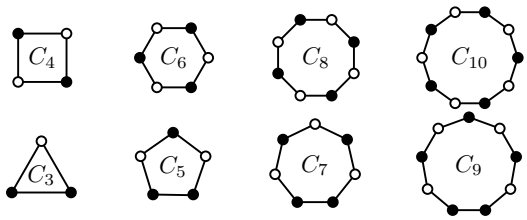
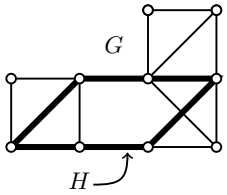


Fig. 16.6 Even cycles are bipartite; odd cycles are not bipartite.

A graph inside a graph is called a *subgraph*.

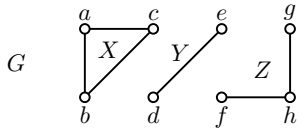
Definition 16.2. A graph H is said to be a **subgraph** of a graph G provided that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

For example, the bolded part of the graph G below is a graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, so H is a subgraph of G. When one graph H is a subgraph of another graph G, we say that G **contains** H, or H **is in** G. In this case, the graph H happens to be a cycle C_6, so we say that G contains a C_6. You can also find a P_{10} in G. You will also find a C_{10}, several K_3's, but no K_4.



A graph G is said to be **connected** if for any two vertices $x, y \in V(G)$, there is a path in G that starts at x and ends at y. A graph that is not connected is said to be **disconnected**. For an example of a disconnected graph, let G be the roadmap of Hawaii, where roads are edges and intersections are vertices. This graph is not connected, because if x and y are vertices on different islands, then G has no subgraph that is a path from x to y.

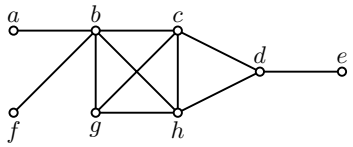
For another example, see the graph G below, with $V(G) = \{a, b, c, d, e, f, g, h\}$, and $E(G) = \{ab, bc, ca, de, fh, gh\}$. This graph is not connected because it has no path joining (say) vertex a to vertex d. The connected "parts" of a disconnected graph are called its *components*. To be precise, a **component** of a graph is a connected subgraph that is not a subgraph of a larger connected subgraph. Thus the above G has components X, Y and Z, as indicated. (Pay careful attention to the wording of the definition of a component. The single edge ab is a connected subgraph of G, but it is not a component because it is a subgraph of the larger connected subgraph X. But the subgraph X *is* a component because it is not a subgraph of any larger *connected* subgraph of G.) Note that a connected graph has just one component; a disconnected graph has more than one component.



Many important types of graphs can be described by the types of subgraphs they have, or don't have. For example, the next section defies the important notion of a *tree*, which is a connected graph that contains no cycles. And we'll see that *bipartite* graphs are precisely the graphs which don't contain any *odd* cycles.

16.2 Vertex-degree and Trees

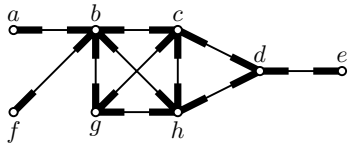
Given a vertex x of a graph G, the **degree** of x, denoted $\deg(x)$, is the number of edges of G that are incident with x. For example, for the graph below, we have $\deg(a) = 1$, $\deg(b) = 5$, $\deg(c) = 4$ and $\deg(d) = 3$.



Imagine adding up the degrees of all the vertices in a graph. In our example above, this would be

$$\deg(a) + \deg(b) + \deg(c) + \deg(d) + \deg(e) + \deg(f) + \deg(g) + \deg(h).$$

This adds up the total number of the bold "spokes" in the below diagram. Because each edge of the graph has exactly two spokes on it, the total number of spokes is twice the number of edges, or $2|E(G)|$.



From this reasoning, you can see that the sum of all the vertex degrees of a graph is twice the number of its edges. We use the notation $\sum_{x \in V(G)} \deg(x)$ to stand for the sum of the degrees of all the vertices of G, the idea being that we add in $\deg(x)$ once for each $x \in V(G)$. Let's summarize this as a fact.

Proposition 16.1. *For any graph G, the sum of the degrees its vertices is twice its number of its edges, that is,*

$$\sum_{x \in V(G)} \deg(x) = 2|E(G)|.$$

In particular, this means that the sum of all the vertex degrees is *even* Now, if the sum of integers is even, then the number of odd terms in the sum must be even. This yields an immediate proposition.

Proposition 16.2. *A graph has an even number of vertices of odd degree.*

For example, the graph on the previous page has 6 vertices of odd degree. Also, $K_{3,4}$ has 4 vertices of odd degree, and K_5 has 0 vertices of odd degree. Try to draw a graph with an odd number of odd-degree vertices. You can't.

Fact 16.1 yields a formula for the average degree of a vertex of a graph, which is the sum of all vertex degrees, divided by the number of vertices. The average vertex degree of a graph G is

$$\frac{\sum_{x \in V(G)} \deg(x)}{|V(G)|} = \frac{2|E(G)|}{|V(G)|}. \tag{16.1}$$

For example, if a graph has 10 vertices and 9 edges, then its average vertex degree is $2 \cdot 9/10 = 18/10 = 1.8$. From this we can infer that the graph has a vertex of degree 0 or 1. (Because if all vertices were degree 2 or greater, the average vertex degree would not be less than 2.)

We now define two fundamental types of graphs: trees and forests.

Definition 16.3. A **tree** is a connected graph that contains no cycles. A **forest** is a graph that contains no cycles.

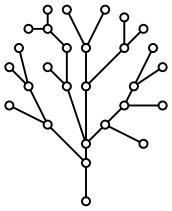


Fig. 16.7 A tree. A **tree** is a connected graph with no cycles.

So forests and trees are graphs without cycles. The difference between the two is that a tree must be connected, but a forest need not be. Every tree is a forest, but a forest is not a tree unless it is connected. Figure 16.7 shows a typical tree, while Figure 16.8 shows a forest. Note that every component of a forest is a tree.

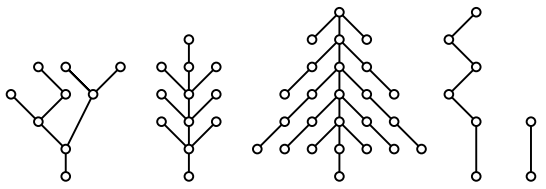


Fig. 16.8 A forest. A **forest** is a graph with no cycles. Every component of a forest is a tree.

For other examples, note that any path P_n is a tree. Also, any complete bipartite graph $K_{1,n}$ (where one partite set has size 1) is a tree.

In every tree we've seen so far, the number of edges is the number of vertices, minus 1. For example, P_n has n vertices and $n-1$ edges. This is a general principle.

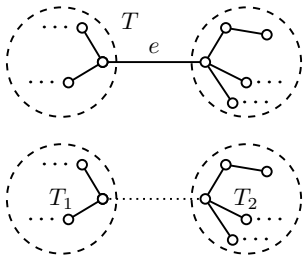
Proposition 16.3. *A tree with n vertices has $n-1$ edges.*

Proof. This theorem asserts that for any $n \in \mathbb{N}$, the following statement is true: S_n : *A tree with n vertices has $n-1$ edges.* We use strong induction to prove this.

For the basis step, observe that if a tree has $n = 1$ vertex, then it has no edges. Thus it has $n - 1 = 0$ edges, so the theorem is true when $n = 1$.

For the inductive step, take any $n > 1$. We must prove $(S_1 \wedge S_2 \wedge \cdots \wedge S_{n-1}) \Rightarrow S_n$. In words, we must show that if it is true that any tree with $k < n$ vertices has $k-1$ edges, then any tree with n vertices has $n-1$ edges. We will use direct proof.

Suppose that for $k < n$, any tree with k vertices has $k-1$ edges. Now let T be a tree with n vertices. Single out an edge of T and label it e, as shown below.



Now remove the edge e from T, but leave the two endpoints of e. This leaves two smaller trees that we call T_1 and T_2. Let's say T_1 has x vertices and T_2 has y vertices. As each of these two smaller trees has fewer than n vertices, our inductive hypothesis guarantees that T_1 has $x-1$ edges, and T_2 has $y-1$ edges. Think about our original tree T. It has $x+y$ vertices. It has $x-1$ edges that belong to T_1 and $y-1$ edges that belong to T_2, *plus* it has the additional edge e that belongs to neither T_1 nor T_2. Thus, all together, the number of edges that T has is $(x-1)+(y-1)+1 = (x+y)-1$. In other words, T has one fewer edges than it has vertices. Thus it has $n-1$ edges.

It follows by strong induction that a tree with n vertices has $n-1$ edges. □

Corollary 16.1. *A forest with n vertices and c components has $n-c$ edges.*

Proof. Say G is a forest with n vertices and c components $G_1, G_2, \ldots, G_c$. Each component G_i is a tree, so it has $|V(G_i)|-1$ edges. Then the total number of edges of G is

$$\sum_{i=1}^{c} \big(|V(G_i)|-1\big) \;=\; \left(\sum_{i=1}^{c} |V(G_i)|\right) - c \;=\; n-c. \qquad \square$$

For example, the forest in Figure 16.8 has 54 vertices and 5 components, so it must have $54 - 5 = 49$ edges, which a count verifies.

One interesting thing about trees is that their average vertex degree is quite low. Equation (16.1) and Proposition 16.3 imply that a tree with n vertices has average degree $\frac{2|E(G)|}{|V(G)|} = \frac{2(n-1)}{n} = \frac{2n-2}{n} = 2 - \frac{2}{n}$. This is less than 2. Consequently every tree must have at least one vertex of degree 0 or 1. But if a tree has more than one vertex, it can't have any vertices of degree 0, because it is connected. This reasoning yields a lemma.

Lemma 16.1. *A tree with more than one vertex has a vertex of degree 1.*

Actually, you can prove a stronger result: Any tree with more than one vertex has at least *two* vertices of degree 1 (Exercise 16.1).

Lemma 16.1 can be especially useful when you are using induction to prove something about trees. The next proof highlights this point.

Proposition 16.4. *Every forest is bipartite.*

Proof. Since the components of a forest are trees, it suffices to prove that every tree is bipartite. To prove this we use induction on the number n of vertices.

For the basis step, note that a tree with 1 vertex is K_1, which is bipartite. Thus the proposition is true for $n = 1$.

Now let $n > 2$, and assume that every tree with fewer than n vertices is bipartite. From this we need to show that any tree with n vertices is bipartite.

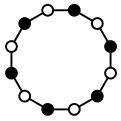
So say T is a tree with $n > 2$ vertices. Lemma 16.1 implies that T has a vertex x of degree 1. Let xy be the sole edge of T with endpoint x. Make a new tree T' by removing x and xy from T. That is, $V(T') = V(T) - \{x\}$ and $E(T') = E(T) - \{xy\}$. But T' has fewer vertices than does T, so T' is bipartite. Color its vertices black and white, in such a way that any edge joins vertices of different colors. Now reconstruct T by adding the edge xy back, and give x the color that is different from y's color.

The vertices of T have now been given two colors, and every edge has differently colored endpoints. Thus T is bipartite. □

Theorem 16.1. *A graph is bipartite if and only if it contains no odd cycle.*

Proof. This is an if and only if theorem, so the proof involves two implications.

First we prove that if a graph G is bipartite, then it has no cycle of odd length. We use direct proof. Let G be bipartite, and let C be an arbitrary cycle in G. We must show that C has *even* length. As G is bipartite, its vertices can be colored black and white, such that the endpoints of each edge have different colors. So the vertices of C alternate black, white, black, white, etc. Thus C has as many white vertices as black vertices, so C is even.



Conversely, we need to show that if a graph has no odd cycles, then it is bipartite. We will prove this using proof by smallest counterexample.

Suppose the statement is false, which means that there exist non-bipartite graphs with no odd cycles. Among all such graphs, let G be one with the fewest number of edges. So G has no odd cycles and is non-bipartite, but any graph with fewer edges and no odd cycles *is* bipartite. Below we will see that this leads to a contradiction.

Note that G is not a forest, because forests are bipartite, by Proposition 16.4. This means that G must have at least one cycle, and because G has no odd cycles, it has an even cycle. Take an even cycle of length $2n$ in G, with vertices $x_1, x_2, x_3, \ldots, x_{2n}$ and edges $x_1x_2,\ x_2x_3,\ x_3x_4,\ \ldots,\ x_{2n-1}x_{2n},\ x_{2n}x_1$. Let G' be the graph G with the edge $x_{2n}x_1$ removed. That is, G' is the graph with $V(G') = V(G)$ and $E(G') = E(G) - \{x_{2n}x_1\}$. Then G' has fewer edges than G, and G' certainly has no odd cycle, because G has none. Therefore G' is bipartite, as G is a smallest non-bipartite graph with no odd cycles. Color the vertices of G' black and white, so that no edge of G' joins vertices of the same color. The vertex sequence $x_1, x_2, x_3, \ldots, x_{2n}$ is on a path in G', so these vertices alternate colors. Note that x_1 and x_{2n} have different colors. Adding the edge x_1x_{2n} back yields G with its vertices colored black and white, and with every edge of G having differently colored endpoints. Therefore G is bipartite, which contradicts the fact that it is not bipartite. □

16.3 Colorings and Chromatic Number

Many practical problems can be modeled and solved by assigning colors to the vertices of a graph. Given an integer k, a **k-coloring** of a graph is an assignment of k colors to the vertices of the graph, so that each vertex gets one of the k colors. Often we denote the k colors by the integers $1, 2, 3, \ldots k$. (For example, 1 means red, 2 means blue, 3 is yellow, etc.) Figure 16.9 shows two 3-colorings and a 5-coloring of a graph.

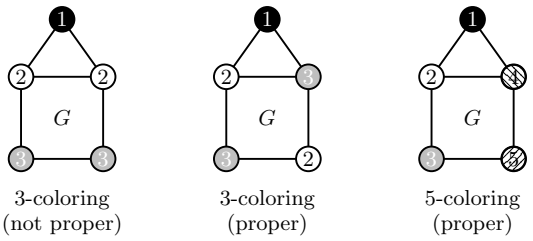


Fig. 16.9 Three different colorings of a graph. The one on the left is not a proper coloring. The other two are proper colorings.

A coloring of a graph is said to be a **proper coloring** if no two adjacent vertices have the same color, that is, if the endpoints of each edge have different colors. The coloring on the left of Figure 16.9 is not proper because there are two edges whose endpoints have the same color. The other two colorings in the figure *are* proper colorings.

Figure 16.9 shows a proper 3-coloring and a proper 5-coloring of G. You will have no trouble finding a proper 4-coloring of G. But you can't draw a proper 2-coloring of this particular graph, because triangle in the graph can't be colored with 2 colors without the endpoints of one of its edges having the same color.

Definition 16.4. The **chromatic number** of a graph, denoted $\chi(G)$, is the smallest integer k for which the graph has a proper k-coloring.

For example, for the graph G in Figure 16.9, we have $\chi(G) = 3$ because the figure shows a proper 3-coloring, but we remarked that no proper 2-coloring (or 1-coloring) exists.

For another example, note that $\chi(K_n) = n$, because we can make a proper coloring by using n colors and giving each of the n vertices its own color. Further, if we colored the n vertices of K_n with fewer than n colors, then this would not be a proper coloring because two (adjacent) vertices would have to have the same color. Thus n is the smallest number of colors required for a proper coloring of K_n, which means $\chi(K_n) = n$.

But for complete bipartite graphs we have $\chi(K_{m,n}) = 2$, because $K_{m,n}$ is bipartite, which means its vertices can be colored with two colors, black and white, with no edge having endpoints of the same color. For the same reason, any bipartite graph that has at least one edge has chromatic number 2.

Concerning the chromatic number of cycles, C_n is bipartite when n is even, so $\chi(C_n) = 2$. For odd cycles, we can obtain a proper coloring with 3 colors as indicated in Figure 16.10, by alternating black and white around the cycle until reaching a final (mismatched) vertex which must be assigned a third color gray.

Therefore a formula for cycles is

$$\chi(C_n) = \begin{cases} 2 \text{ if } n \text{ is even} \\ 3 \text{ if } n \text{ is odd.} \end{cases}$$

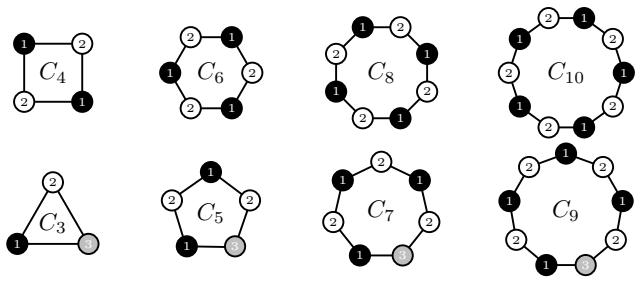


Fig. 16.10 For even cycles, $\chi(C_n) = 2$. For odd cycles, $\chi(C_n) = 3$.

There are many practical optimization problems in fields as diverse as computer science and telecommunications that can be solved by finding the chromatic number of a graph. Here is a simple example involving scheduling.

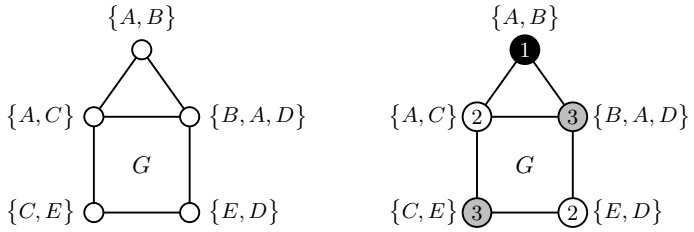
Example 16.1. An office has five employees, call them A, B, C, D and E. They belong to five committees, $\{A, B\}$, $\{A, C\}$, $\{B, A, D\}$, $\{C, E\}$ and $\{E, D\}$ (the sets indicate who's on each committee). One day, each committee has to meet for one hour. Obviously all five committees can't meet during the same hour, because some people belong to several committees and can't be two places at the same time. What is the fewest number of hours needed for all of the committees to meet?

Solution: The simple-minded solution is to block off hours of the day for each committee to meet, one after the other. This takes five hours.

hour	time	committee meeting
1	1:00–2:00	$\{A, B\}$
2	2:00–3:00	$\{A, C\}$
3	3:00–4:00	$\{B, A, D\}$
4	4:00–5:00	$\{C, E\}$
5	5:00–6:00	$\{E, D\}$

But the meetings can be done in fewer than five hours because committees with no members in common can meet at the same time. We need to find the *fewest* number of hours required for all committees to meet.

This problem can be modeled with a graph. Let the vertices be the committees, and connect an edge between two committees whenever they have a person in common, and therefore can't meet at the same time. The graph G is drawn below.



To find an optimal schedule, we can give G a proper coloring, using colors corresponding to hours 1,2,3,..., so that no two adjacent committees meet at the same hour (and using the fewest possible colors). This happens to be the same graph as in Figure 16.9, where $\chi(G) = 3$, and a proper 3-coloring is shown above on the right. Thus the optimal schedule uses only three hours:

hour	time	committee meeting
1	1:00–2:00	$\{A, B\}$
2	2:00–3:00	$\{A, C\}$ and $\{E, D\}$
3	3:00–4:00	$\{C, E\}$ and $\{B, A, D\}$

Though this example is simple, you can imagine much more complicated situations in which the chromatic number would optimize efficiency. ✍

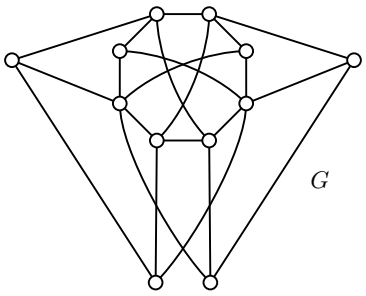
Given that the chromatic number of a graph has the potential to solve real-life problems, we would hope for a formula or algorithm to compute it. Unfortunately, the general problem of finding the chromatic number turns out to be what is called an *NP-complete* problem, meaning—roughly—that the general consensus is that a simple formula or efficient algorithm for the chromatic number of an arbitrary graph is impossible. We will learn about NP-completeness in Chapter **??**. What this means for us now is that computing chromatic numbers is somewhat of an art, and sometimes (for complex graphs) the best we can hope for is an estimate. Here is a proposition that, if cleverly used, gives a lower bound for $\chi(G)$.

Proposition 16.5. *If H is a subgraph of G, then $\chi(H) \leq \chi(G)$.*

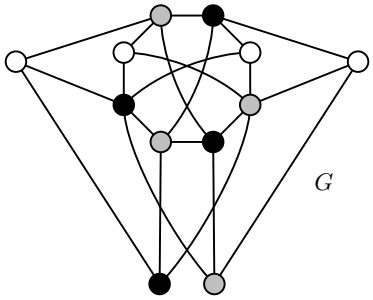
Proof. We use direct proof. Suppose that H is a subgraph of G. By definition of the chromatic number, G has a proper coloring with $\chi(G)$ colors. Because every vertex of H is also a vertex of G, the vertices of H are colored by the coloring of G. And because each edge of H is an edge of G, no two edges of H have the same color. Thus H is properly colored with no more than $\chi(G)$ colors. (Possibly some of the colors of G's vertices do not appear in H's.) It follows that the chromatic number of H is at most $\chi(G)$, that is, $\chi(H) \leq \chi(G)$. □

We can use this proposition help find $\chi(G)$. If we can find a subgraph H of G that is a familiar graph with a known chromatic number (such as K_n or C_n), then we know the chromatic number of the unfamiliar graph G is not smaller than that of the familiar graph H. Thus we can rule out k-colorings of G that use fewer colors than $\chi(H)$. Sometimes this information is enough that a little informed experimentation can zero in on $\chi(G)$.

Example 16.2. Find the chromatic number of the graph drawn below.



Solution. This graph has a 5-cycle C_5 as a subgraph (find it), and $\chi(C_5) = 3$, so Proposition 16.5 says $3 = \chi(C_5) \leq \chi(G)$. Thus $\chi(G)$ cannot be smaller than 3. But trial-and-error gives the following coloring of G with exactly 3 colors.



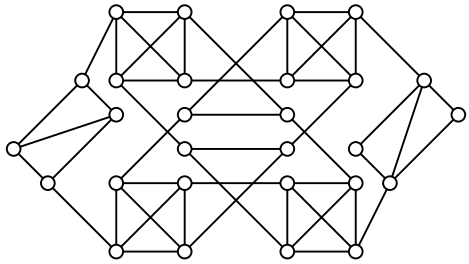
Thus in fact $\chi(G) = 3$. ✍

Proposition 16.5 gives a lower bound on $\chi(G)$, but it gives no information about how much bigger $\chi(G)$ is than $\chi(H)$. The previous example used an additional trial-and-error step to show $\chi(G)$ was exactly $\chi(H)$. For an upper bound, we have Brook's Theorem.

Theorem 16.2. (Brook's Theorem) *Let G be a connected graph that is neither a complete graph nor an odd cycle. If the largest vertex degree of G is Δ, then $\chi(G) \leq \Delta$.*

The proof of Brook's theorem is too long to include here. (But if you have come this far it is within your grasp.) If you enjoy the subject, then perhaps you will encounter a proof of it later, in a first course in graph theory.

Example 16.3. Find the chromatic number of the graph drawn below.



Solution. The graph has a K_4 as a subgraph, and we know $\chi(K_4) = 4$. Thus Proposition 16.5 yields $4 = \chi(K_4) \leq \chi(G)$. But also the largest vertex degree is $\Delta = 4$, so Brook's theorem gives $\chi(G) \leq \Delta = 4$. Consequently $4 \leq \chi(G) \leq 4$, so $\chi(G) = 4$. ✍

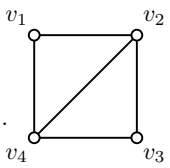
To conclude our brief foray into coloring, let's now consider algorithmic approaches to graph coloring. Actually, as was noted before, deciding if a graph has chromatic number greater than 3 is known to be a hopelessly difficult problem, in general. (We owe our success with the above examples to the fact that the graphs

were small, and carefully chosen to yield a solution under our imperfect techniques.) However, it is possible to write a simple, efficient algorithm that decides whether or not $\chi(G) = 2$. We will focus on that.

The main reason for this investigation is not that our algorithm will be useful or important, but rather that it sets the stage for deeper issues that await us in Chapter **??**, and that it illustrates the idea of an algorithm that solves a graph theory problem.

It will also prompt us to confront the problem of representing a graph as data that an algorithm can read as input. Here is one way to do it. A graph, such as the one to the right, can be modeled as the list of lists

$$\Big(\big(v_1,(v_2,v_4)\big),\ \ \big(v_2,(v_1,v_3,v_4)\big),\ \ \big(v_3,(v_2,v_4)\big),\ \ \big(v_4,(v_1,v_2,v_4)\big)\Big).$$

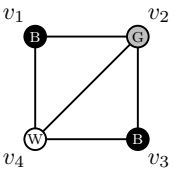


The first entry $\big(v_1,(v_2,v_4)\big)$ encodes the fact that v_1 is a vertex, and it is adjacent to v_2 and v_4. We say v_2 and v_4 are the **neighbors** of v_1. The second entry $\big(v_2,(v_1,v_3,v_4)\big)$ encodes the fact that v_2 is a vertex that has neighbors v_1, v_3 and v_4, etc. In general, a graph with n vertices $v_1, v_2, \ldots, v_n$ can be encoded as

$$G = \Big(\big(v_1,N_1\big),\ \ \big(v_2,N_2\big)\big),\ \ \big(v_3,N_3\big),\ \ \ldots\ \ \big(v_n,N_n\big)\big)\Big), \tag{16.2}$$

where each N_i is a list of neighbors of v_i. There is some redundancy in this scheme, as $v_k \in N_\ell$ if and only if $v_\ell \in N_k$. (That is, v_ℓ is a neighbor of v_k if and only if v_k is a neighbor of v_ℓ, so why list it twice?) However, this scheme has the advantage of granting immediate access to the neighbors of a vertex.

We can represent a coloring of the graph (16.2) with a list of colors $C = (c_1, c_2, \ldots, c_n)$, meaning v_1 gets color c_1, and v_2 gets color c_2, etc. For example, the coloring on the right is described by the list (B,G,B,W). This is the way that our two-coloring algorithm will represent a coloring. Actually, even though its goal is to 2-color a graph, it allows for three colors: black (B), white (W), and "uncolored" (U). The algorithm will start by assigning almost all vertices the color U, then change the colors step by step until a 2-coloring is found.



The algorithm uses the fact that if a connected graph can be properly 2-colored, then as soon as we color one vertex, the entire proper coloring uniquely determined; that is, the colors of all the other vertices are forced. Here's why: Suppose we color the vertex v_1 black. Then all its neighbors must be colored *white*. And all *their* neighbors must be colored *black*, etc., until all vertices are reached and assigned colors. So if the graph has a proper 2-coloring, this process will find it. And if there is no proper coloring, then a some point there will be a collision; like a white vertex causing its neighbors to be black, when one neighbor was previously colored white.

So the algorithm starts by giving v_1 the color B (black), and all other vertices the color U (uncolored). Then a loop passes through all vertices v_i, and whenever

a colored one is found, any uncolored neighbor v_k is assigned the opposite color. In this way, either all the vertices are eventually colored (resulting in a proper 2-coloring) or a v_i is found to be the same color as its neighbor, in which case we do not have a proper 2-coloring, so $\chi(G) > 2$.

The somewhat non-standard command "**foreach** $v_k \in N_i$ **do**" means to issue the commands that follow for every neighbor v_k of v_i.

Algorithm 13: For 2-coloring a graph

```
Input: A connected graph G = ((v_1, N_1), (v_2, N_2), ..., (v_n, N_n)) with
       n ≥ 2
Output: "NO" if χ(G) > 2; otherwise proper 2-coloring C = (c_1, c_2, ..., c_n)
begin
    C := (B,U,U,..., U) ..... v_1 is black; all other v_i are uncolored
    colored := 1 ..... number of colored vertices; so far 1 vertex colored
    failure = NO ..... havent't failed yet
    while colored < n do
        for i := 1 to n do
            if c_i ≠ U then
                foreach v_k ∈ N_i do
                    if c_k = U then
                        if c_i = B then
                            c_k := W
                        else
                            c_k := B
                        end
                        colored := colored + 1
                        foreach v_ℓ ∈ N_k do
                            if c_ℓ = c_k then
                                colored := n
                                failure := YES
                            end
                        end
                    end
                end
            end
        end
    end
    if failure = YES then
        output NO ..... no 2-coloring of G exists
    else
        output C ..... This is a 2-coloring of G
    end
end
```

Exercises for Chapter 16

1. Prove that a tree with more than one vertex has at least two vertices of degree 1.

2. A graph G is bipartite if and only if there is a homomorphism $G \to K_2$.

3. Suppose G is a connected graph. Prove that G is **not** bipartite if and only if there is an integer n such that for any pair $x, y \in V(G)$ and any integer $k \geq n$, there is an x, y-walk of length k.

4. A graph is bipartite if and only if for any pair $x, y \in V(G)$ any two x, y-walks have the same parity (i.e. both of even length, or both of odd length).