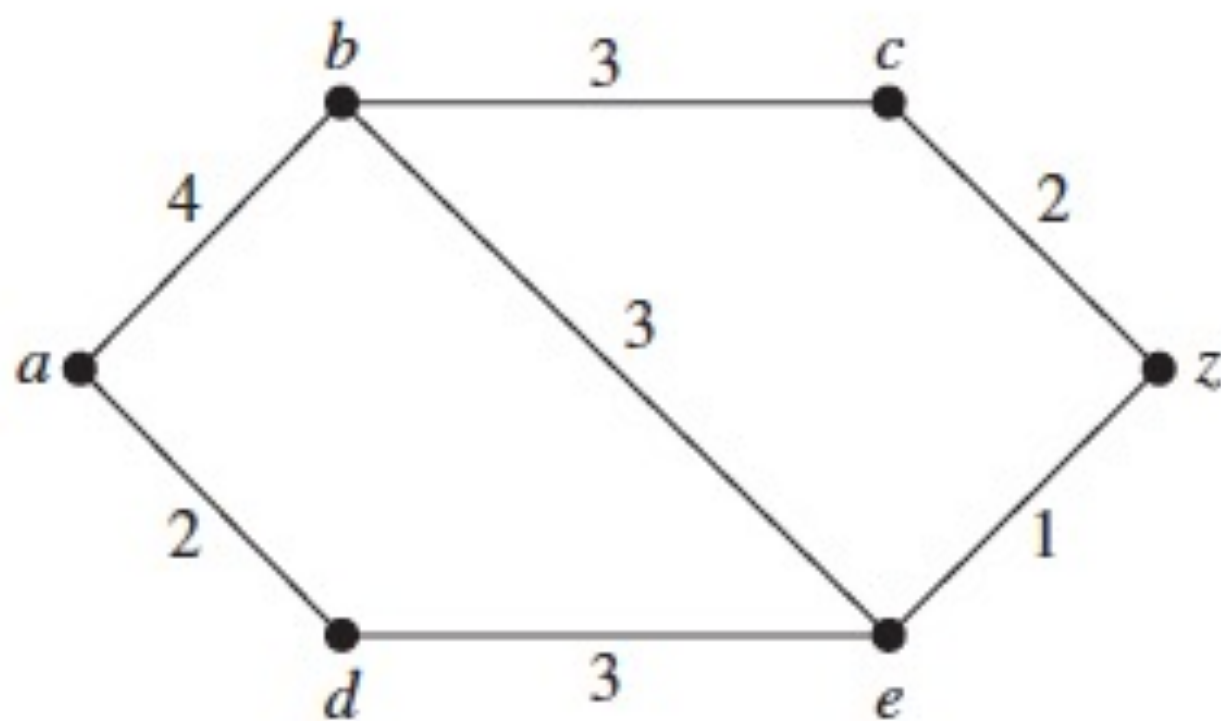


# Weighted Graphs and Shortest Path Algorithm



**FIGURE 3** A Weighted Simple Graph.


What is the length of a shortest path between  $a$  and  $z$  in the weighted graph shown in Figure 3?

**Solution:** Although a shortest path is easily found by inspection, we will develop some ideas useful in understanding Dijkstra's algorithm. We will solve this problem by finding the length of a shortest path from  $a$  to successive vertices, until  $z$  is reached.

The only paths starting at  $a$  that contain no vertex other than  $a$  are formed by adding an edge that has  $a$  as one endpoint. These paths have only one edge. They are  $a, b$  of length 4 and  $a, d$  of length 2. It follows that  $d$  is the closest vertex to  $a$ , and the shortest path from  $a$  to  $d$  has length 2.

We can find the second closest vertex by examining all paths that begin with the shortest path from  $a$  to a vertex in the set  $\{a, d\}$ , followed by an edge that has one endpoint in  $\{a, d\}$  and its other endpoint not in this set. There are two such paths to consider,  $a, d, e$  of length 7 and  $a, b$  of length 4. Hence, the second closest vertex to  $a$  is  $b$  and the shortest path from  $a$  to  $b$  has length 4.

To find the third closest vertex to  $a$ , we need examine only the paths that begin with the shortest path from  $a$  to a vertex in the set  $\{a, d, b\}$ , followed by an edge that has one endpoint in the set  $\{a, d, b\}$  and its other endpoint not in this set. There are three such paths,  $a, b, c$  of length 7,  $a, b, e$  of length 7, and  $a, d, e$  of length 5. Because the shortest of these paths is  $a, d, e$ , the third closest vertex to  $a$  is  $e$  and the length of the shortest path from  $a$  to  $e$  is 5.

To find the fourth closest vertex to  $a$ , we need examine only the paths that begin with the shortest path from  $a$  to a vertex in the set  $\{a, d, b, e\}$ , followed by an edge that has one endpoint in the set  $\{a, d, b, e\}$  and its other endpoint not in this set. There are two such paths,  $a, b, c$  of length 7 and  $a, d, e, z$  of length 6. Because the shorter of these paths is  $a, d, e, z$ , the fourth closest vertex to  $a$  is  $z$  and the length of the shortest path from  $a$  to  $z$  is 6. 

Example 1 illustrates the general principles used in Dijkstra's algorithm. Note that a shortest path from  $a$  to  $z$  could have been found by a brute force approach by examining the length of every path from  $a$  to  $z$ . However, this brute force approach is impractical for humans and even for computers for graphs with a large number of edges.



We will now consider the general problem of finding the length of a shortest path between  $a$  and  $z$  in an undirected connected simple weighted graph. Dijkstra's algorithm proceeds by finding the length of a shortest path from  $a$  to a first vertex, the length of a shortest path from  $a$  to a second vertex, and so on, until the length of a shortest path from  $a$  to  $z$  is found. As a side benefit, this algorithm is easily extended to find the length of the shortest path from  $a$  to all other vertices of the graph, and not just to  $z$ .

The algorithm relies on a series of iterations. A distinguished set of vertices is constructed by adding one vertex at each iteration. A labeling procedure is carried out at each iteration. In this labeling procedure, a vertex  $w$  is labeled with the length of a shortest path from  $a$  to  $w$  that contains only vertices already in the distinguished set. The vertex added to the distinguished set is one with a minimal label among those vertices not already in the set.

We now give the details of Dijkstra's algorithm. It begins by labeling  $a$  with 0 and the other vertices with  $\infty$ . We use the notation  $L_0(a) = 0$  and  $L_0(v) = \infty$  for these labels before any iterations have taken place (the subscript 0 stands for the "0th" iteration). These labels are the lengths of shortest paths from  $a$  to the vertices, where the paths contain only the vertex  $a$ . (Because no path from  $a$  to a vertex different from  $a$  exists,  $\infty$  is the length of a shortest path between  $a$  and this vertex.)

Dijkstra's algorithm proceeds by forming a distinguished set of vertices. Let  $S_k$  denote this set after  $k$  iterations of the labeling procedure. We begin with  $S_0 = \emptyset$ . The set  $S_k$  is formed from  $S_{k-1}$  by adding a vertex  $u$  not in  $S_{k-1}$  with the smallest label.

Once  $u$  is added to  $S_k$ , we update the labels of all vertices not in  $S_k$ , so that  $L_k(v)$ , the label of the vertex  $v$  at the  $k$ th stage, is the length of a shortest path from  $a$  to  $v$  that contains vertices only in  $S_k$  (that is, vertices that were already in the distinguished set together with  $u$ ). Note that the way we choose the vertex  $u$  to add to  $S_k$  at each step is an optimal choice at each step, making this a greedy algorithm. (We will prove shortly that this greedy algorithm always produces an optimal solution.)



Let  $v$  be a vertex not in  $S_k$ . To update the label of  $v$ , note that  $L_k(v)$  is the length of a shortest path from  $a$  to  $v$  containing only vertices in  $S_k$ . The updating can be carried out efficiently when this observation is used: A shortest path from  $a$  to  $v$  containing only elements of  $S_k$  is either a shortest path from  $a$  to  $v$  that contains only elements of  $S_{k-1}$  (that is, the distinguished vertices not including  $u$ ), or it is a shortest path from  $a$  to  $u$  at the  $(k - 1)$ st stage with the edge  $\{u, v\}$  added. In other words,

$$L_k(a, v) = \min\{L_{k-1}(a, v), L_{k-1}(a, u) + w(u, v)\},$$

where  $w(u, v)$  is the length of the edge with  $u$  and  $v$  as endpoints. This procedure is iterated by successively adding vertices to the distinguished set until  $z$  is added. When  $z$  is added to the distinguished set, its label is the length of a shortest path from  $a$  to  $z$ .

### ALGORITHM 1 Dijkstra's Algorithm.

**procedure** *Dijkstra*( $G$ : weighted connected simple graph, with  
all weights positive)

{ $G$  has vertices  $a = v_0, v_1, \dots, v_n = z$  and lengths  $w(v_i, v_j)$   
where  $w(v_i, v_j) = \infty$  if  $\{v_i, v_j\}$  is not an edge in  $G$ }

**for**  $i := 1$  **to**  $n$

$L(v_i) := \infty$

$L(a) := 0$

$S := \emptyset$

{the labels are now initialized so that the label of  $a$  is 0 and all  
other labels are  $\infty$ , and  $S$  is the empty set}

**while**  $z \notin S$

$u :=$  a vertex not in  $S$  with  $L(u)$  minimal

$S := S \cup \{u\}$

**for** all vertices  $v$  not in  $S$


**if**  $L(u) + w(u, v) < L(v)$  **then**  $L(v) := L(u) + w(u, v)$

        {this adds a vertex to  $S$  with minimal label and updates the  
        labels of vertices not in  $S$ }

**return**  $L(z)$  { $L(z)$  = length of a shortest path from  $a$  to  $z$ }



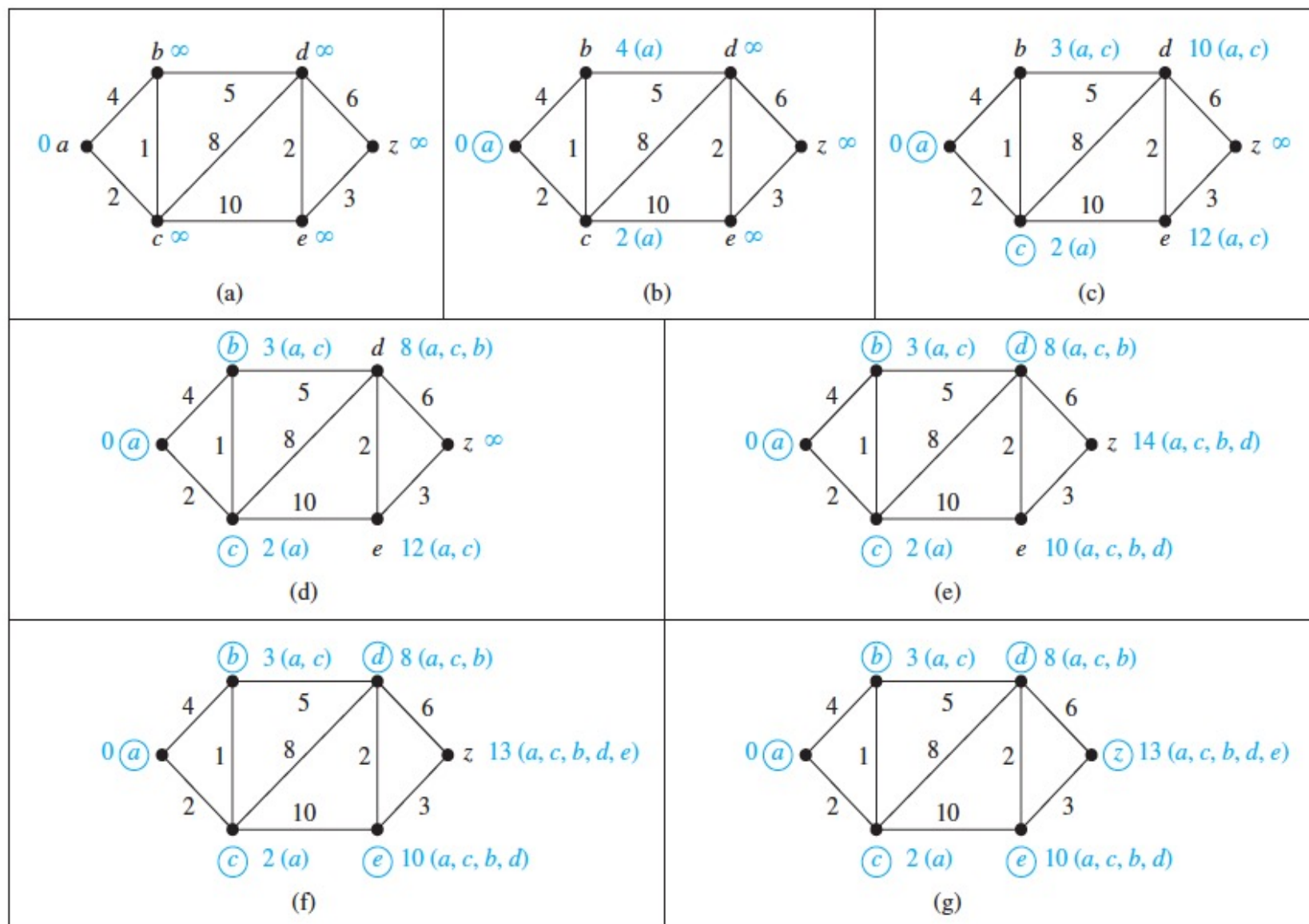
Use Dijkstra's algorithm to find the length of a shortest path between the vertices  $a$  and  $z$  in the weighted graph displayed in Figure 4(a).

*Solution:* The steps used by Dijkstra's algorithm to find a shortest path between  $a$  and  $z$  are shown in Figure 4. At each iteration of the algorithm the vertices of the set  $S_k$  are circled. A shortest path from  $a$  to each vertex containing only vertices in  $S_k$  is indicated for each iteration. The algorithm terminates when  $z$  is circled. We find that a shortest path from  $a$  to  $z$  is  $a, c, b, d, e, z$ , with length 13. 

*Remark:* In performing Dijkstra's algorithm it is sometimes more convenient to keep track of labels of vertices in each step using a table instead of redrawing the graph for each step.

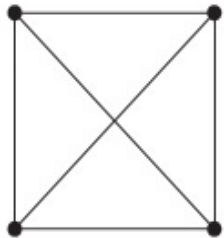
## THEOREM 1

Dijkstra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.

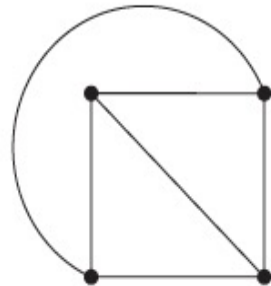


**FIGURE 4** Using Dijkstra's Algorithm to Find a Shortest Path from  $a$  to  $z$ .

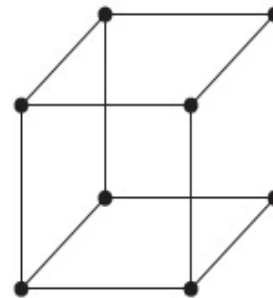
# Planar Graphs



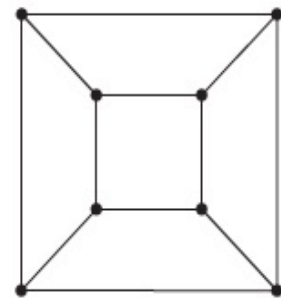
**FIGURE 2** The Graph  $K_4$ .



**FIGURE 3**  $K_4$  Drawn with No Crossings.



**FIGURE 4** The Graph  $Q_3$ .



**FIGURE 5** A Planar Representation of  $Q_3$ .

## DEFINITION 1

A graph is called *planar* if it can be drawn in the plane without any edges crossing (where a crossing of edges is the intersection of the lines or arcs representing them at a point other than their common endpoint). Such a drawing is called a *planar representation* of the graph.

A graph may be planar even if it is usually drawn with crossings, because it may be possible to draw it in a different way without crossings.

**EXAMPLE 1** Is  $K_4$  (shown in Figure 2 with two edges crossing) planar?

*Solution:*  $K_4$  is planar because it can be drawn without crossings, as shown in Figure 3. ◀



## Euler's Formula

A planar representation of a graph splits the plane into **regions**, including an unbounded region. For instance, the planar representation of the graph shown in Figure 8 splits the plane into six regions. These are labeled in the figure. Euler showed that all planar representations of a graph split the plane into the same number of regions. He accomplished this by finding a relationship among the number of regions, the number of vertices, and the number of edges of a planar graph.

### THEOREM 1

**EULER'S FORMULA** Let  $G$  be a connected planar simple graph with  $e$  edges and  $v$  vertices. Let  $r$  be the number of regions in a planar representation of  $G$ . Then  $r = e - v + 2$ .

### COROLLARY 1

If  $G$  is a connected planar simple graph with  $e$  edges and  $v$  vertices, where  $v \geq 3$ , then  $e \leq 3v - 6$ .

Before we prove Corollary 1 we will use it to prove the following useful result.

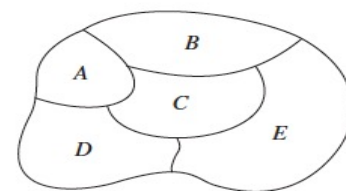
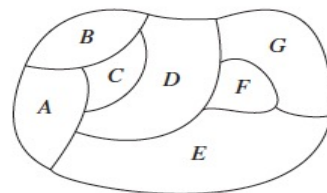
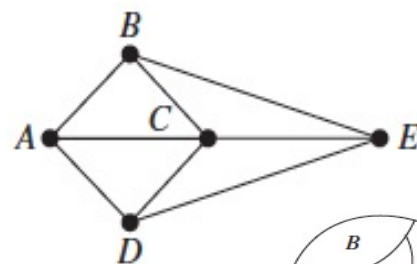
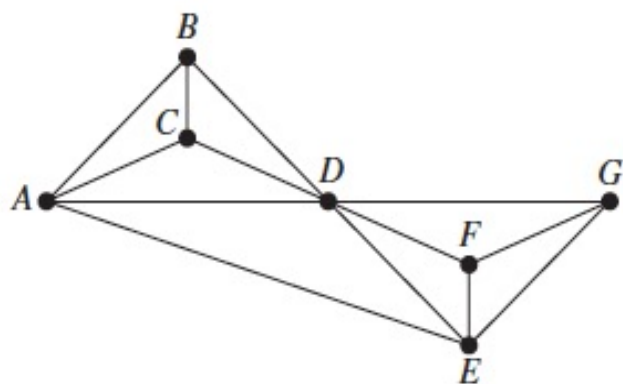
### COROLLARY 2

If  $G$  is a connected planar simple graph, then  $G$  has a vertex of degree not exceeding five.

Each map in the plane can be represented by a graph. To set up this correspondence, each region of the map is represented by a vertex. Edges connect two vertices if the regions represented by these vertices have a common border. Two regions that touch at only one point are not considered adjacent. The resulting graph is called the **dual graph** of the map. By the way in which dual graphs of maps are constructed, it is clear that any map in the plane has a planar dual graph. Figure 2 displays the dual graphs that correspond to the maps shown in Figure 1.

The problem of coloring the regions of a map is equivalent to the problem of coloring the vertices of the dual graph so that no two adjacent vertices in this graph have the same color. We now define a graph coloring.

A *coloring* of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.



**FIGURE 2** Dual Graphs of the Maps in Figure 1.

**FIGURE 1** Two Maps.



## DEFINITION 2

The *chromatic number* of a graph is the least number of colors needed for a coloring of this graph. The chromatic number of a graph  $G$  is denoted by  $\chi(G)$ . (Here  $\chi$  is the Greek letter *chi*.)

Note that asking for the chromatic number of a planar graph is the same as asking for the minimum number of colors required to color a planar map so that no two adjacent regions are assigned the same color. This question has been studied for more than 100 years. The answer is provided by one of the most famous theorems in mathematics.

## THEOREM 1

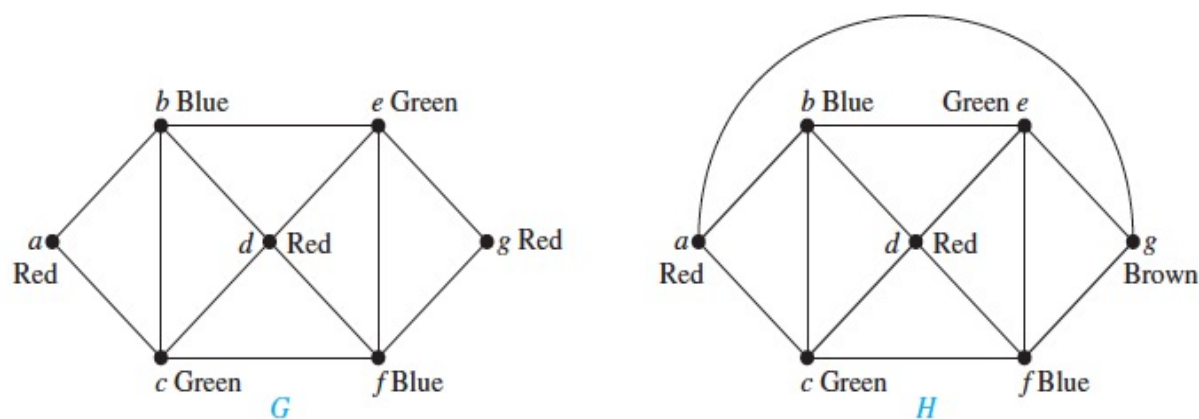
**THE FOUR COLOR THEOREM** The chromatic number of a planar graph is no greater than four.

Two things are required to show that the chromatic number of a graph is  $k$ . First, we must show that the graph can be colored with  $k$  colors. This can be done by constructing such a coloring. Second, we must show that the graph cannot be colored using fewer than  $k$  colors.

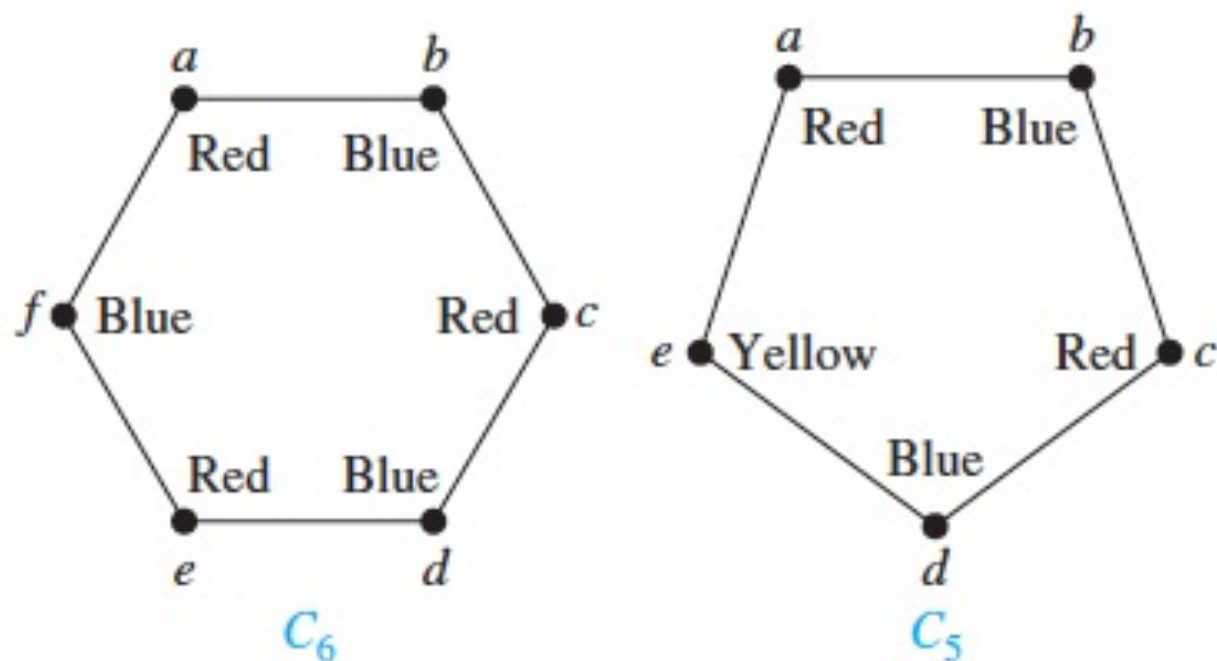
What are the chromatic numbers of the graphs  $G$  and  $H$  shown in Figure 3?

**Solution:** The chromatic number of  $G$  is at least three, because the vertices  $a$ ,  $b$ , and  $c$  must be assigned different colors. To see if  $G$  can be colored with three colors, assign red to  $a$ , blue to  $b$ , and green to  $c$ . Then,  $d$  can (and must) be colored red because it is adjacent to  $b$  and  $c$ . Furthermore,  $e$  can (and must) be colored green because it is adjacent only to vertices colored red and blue, and  $f$  can (and must) be colored blue because it is adjacent only to vertices colored red and green. Finally,  $g$  can (and must) be colored red because it is adjacent only to vertices colored blue and green. This produces a coloring of  $G$  using exactly three colors. Figure 4 displays such a coloring.

The graph  $H$  is made up of the graph  $G$  with an edge connecting  $a$  and  $g$ . Any attempt to color  $H$  using three colors must follow the same reasoning as that used to color  $G$ , except at the last stage, when all vertices other than  $g$  have been colored. Then, because  $g$  is adjacent (in  $H$ ) to vertices colored red, blue, and green, a fourth color, say brown, needs to be used. Hence,  $H$  has a chromatic number equal to 4. A coloring of  $H$  is shown in Figure 4. ◀



**FIGURE 4** Colorings of the Graphs  $G$  and  $H$ .




**FIGURE 7** Colorings of  $C_5$  and  $C_6$ .



In general, two colors are needed to color  $C_n$  when  $n$  is even. To construct such a coloring, simply pick a vertex and color it red. Proceed around the graph in a clockwise direction (using a planar representation of the graph) coloring the second vertex blue, the third vertex red, and so on. The  $n$ th vertex can be colored blue, because the two vertices adjacent to it, namely the  $(n - 1)$ st and the first vertices, are both colored red.

When  $n$  is odd and  $n > 1$ , the chromatic number of  $C_n$  is 3. To see this, pick an initial vertex. To use only two colors, it is necessary to alternate colors as the graph is traversed in a clockwise direction. However, the  $n$ th vertex reached is adjacent to two vertices of different colors, namely, the first and  $(n - 1)$ st. Hence, a third color must be used.

We have shown that  $\chi(C_n) = 2$  if  $n$  is an even positive integer with  $n \geq 4$  and  $\chi(C_n) = 3$  if  $n$  is an odd positive integer with  $n \geq 3$ . 

# Minimum Spanning Tree

A wide variety of problems are solved by finding a spanning tree in a weighted graph such that the sum of the weights of the edges in the tree is a minimum.

A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

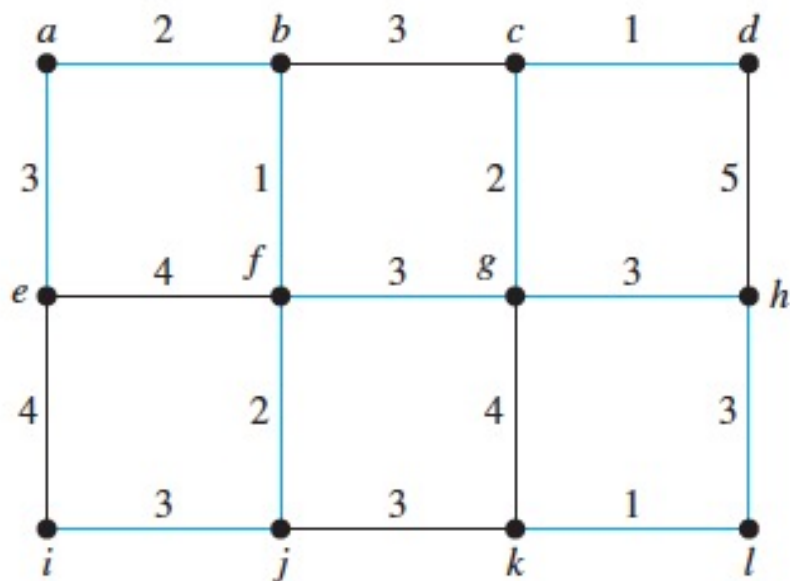
The first algorithm that we will discuss was originally discovered by the Czech mathematician Vojtěch Jarník in 1930, who described it in a paper in an obscure Czech journal. The algorithm became well known when it was rediscovered in 1957 by Robert Prim. Because of this, it is known as **Prim's algorithm** (and sometimes as the **Prim-Jarník algorithm**). Begin by choosing any edge with smallest weight, putting it into the spanning tree. Successively add to the tree edges of minimum weight that are incident to a vertex already in the tree, never forming a simple circuit with those edges already in the tree. Stop when  $n - 1$  edges have been added.



### ALGORITHM 1 Prim's Algorithm.

```
procedure Prim( $G$ : weighted connected undirected graph with  $n$  vertices)
 $T :=$  a minimum-weight edge
for  $i := 1$  to  $n - 2$ 
     $e :=$  an edge of minimum weight incident to a vertex in  $T$  and not forming a
        simple circuit in  $T$  if added to  $T$ 
     $T := T$  with  $e$  added
return  $T$  { $T$  is a minimum spanning tree of  $G$ }
```

Note that the choice of an edge to add at a stage of the algorithm is not determined when there is more than one edge with the same weight that satisfies the appropriate criteria. We need to order the edges to make the choices deterministic. We will not worry about this in the remainder of the section. Also note that there may be more than one minimum spanning tree for a given connected weighted simple graph. (See Exercise 9.) Examples 1 and 2 illustrate how Prim's algorithm is used.



(a)

Choice	Edge	Weight
1	{b, f}	1
2	{a, b}	2
3	{f, j}	2
4	{a, e}	3
5	{i, j}	3
6	{f, g}	3
7	{c, g}	2
8	{c, d}	1
9	{g, h}	3
10	{h, l}	3
11	{k, l}	1
Total:		24

(b)

**FIGURE 4** A Minimum Spanning Tree Produced Using Prim's Algorithm.

The second algorithm we will discuss was discovered by Joseph Kruskal in 1956, although the basic ideas it uses were described much earlier. To carry out **Kruskal's algorithm**, choose an edge in the graph with minimum weight.

Successively add edges with minimum weight that do not form a simple circuit with those edges already chosen. Stop after  $n - 1$  edges have been selected.


#### ALGORITHM 2 Kruskal's Algorithm.

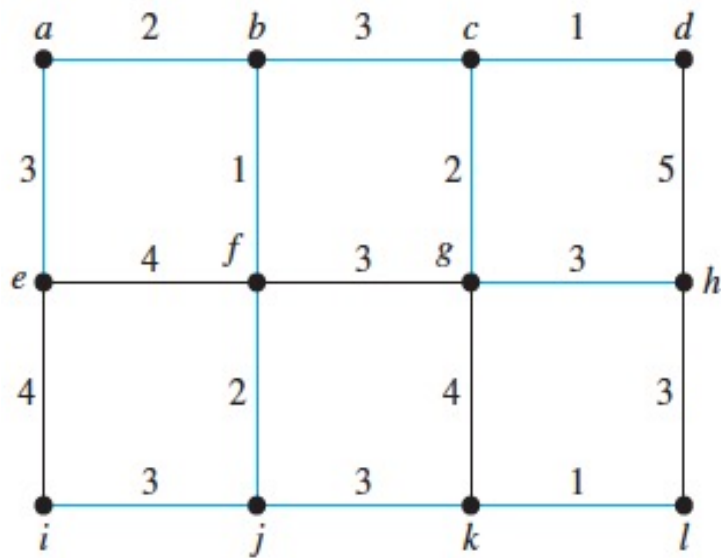
```
procedure Kruskal( $G$ : weighted connected undirected graph with  $n$  vertices)
 $T :=$  empty graph
for  $i := 1$  to  $n - 1$ 
     $e :=$  any edge in  $G$  with smallest weight that does not form a simple circuit
        when added to  $T$ 
     $T := T$  with  $e$  added
return  $T$  { $T$  is a minimum spanning tree of  $G$ }
```



The reader should note the difference between Prim's and Kruskal's algorithms. In Prim's algorithm edges of minimum weight that are incident to a vertex already in the tree, and not forming a circuit, are chosen; whereas in Kruskal's algorithm edges of minimum weight that are not necessarily incident to a vertex already in the tree, and that do not form a circuit, are chosen. Note that as in Prim's algorithm, if the edges are not ordered, there may be more than one choice for the edge to add at a stage of this procedure. Consequently, the edges need to be ordered for the procedure to be deterministic. Example 3 illustrates how Kruskal's algorithm is used.

Use Kruskal's algorithm to find a minimum spanning tree in the weighted graph shown in Figure 3.

*Solution:* A minimum spanning tree and the choices of edges at each stage of Kruskal's algorithm are shown in Figure 5. 



(a)

Choice	Edge	Weight
1	$\{c, d\}$	1
2	$\{k, l\}$	1
3	$\{b, f\}$	1
4	$\{c, g\}$	2
5	$\{a, b\}$	2
6	$\{f, j\}$	2
7	$\{b, c\}$	3
8	$\{j, k\}$	3
9	$\{g, h\}$	3
10	$\{i, j\}$	3
11	$\{a, e\}$	3
Total:		24

(b)

**FIGURE 5** A Minimum Spanning Tree Produced by Kruskal's Algorithm.