# 7.1 Networks and Trees

## Networks

A **network** is a just another name for a *connected graph*. (In the context of networks, vertices are often called *nodes* and edges are called *links*.) Most of the networks we will consider in this chapter will be *simple networks* (i.e., without loops or multiple edges), but we do not make this a requirement. In some applications, a network can have loops, multiple edges, or both.

Our first example illustrates how social networks evolve. Social networks—such as Facebook, Twitter, and Instagram—are networks that connect people through some sort of social relationship—friendship, business, etc. The example is small, but you can imagine the same idea working on a much larger scale.

### EXAMPLE 7.1 SOCIAL NETWORKS

Imagine 15 students (named *A* through *O*) enrolled in a very popular seminar called *The Mathematics of Social Networks*. One of the goals in a small seminar like this one is to get the students to connect with each other and exchange ideas as much as possible, in other words, to "network." For the purposes of this example we will say that two students in the seminar have *connected* if they have exchanged phone numbers or email addresses (presumably for the purposes of intellectual exchange, but we won't really dwell into their reasons for doing so). We can best visualize the interconnections among students in the seminar by means of a *connections graph*: the vertices (nodes) of the graph are the students, and pairs of students are linked by an edge if they have connected according to our definition of the term.

Figure 7-1(a) shows one possible version of the connections graph. In this scenario the graph is not a network but rather three separate, disconnected networks. Looking at this graph would not make the instructor happy. Figure 7-1(b) shows the new connections graph after two additional connections have been added: *EG* and *CH*. Now the graph becomes a true social network. The instructor is much happier—all students can connect, either directly or through intermediaries.

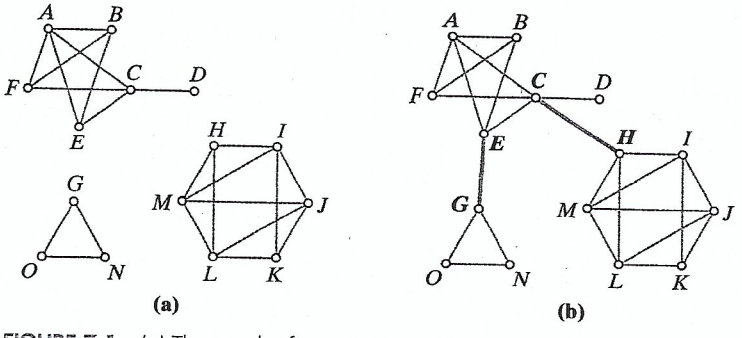



FIGURE 7-1 (a) The graph of connections with three separate components. (b) The graph of connections becomes a network.

Because a network is a connected graph, there are paths going from any vertex to any other vertex—at least one but usually many more. We are usually interested in the *shortest path* connecting a pair of vertices—in other words, a path whose length is as small as possible. We will call the length of a shortest path joining two vertices in a network the **degree of separation** of the two vertices.

### EXAMPLE 7.2 DEGREES OF SEPARATION

Consider again the network in Fig. 7-1(b). Let's look at degrees of separation between different pairs of students.

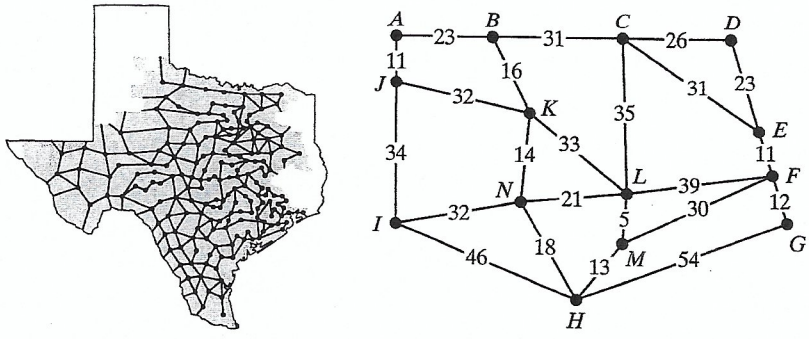
- $A$ and $B$ are joined by an edge, so we will say there is *one* degree of separation between them.
- $A$ and $G$ are not directly connected, but there is a path of length 2 connecting them: $A, E, G$. We will say that there are *two* degrees of separation between $A$ and $G$.
- $A$ and $K$ are connected by the path $A, C, H, M, L, K$ of length 5, but this is not as short as possible. The path $A, C, H, L, K$ has length 4. Can we do even better? No. There is a second path of length 4 ($A, C, H, I, K$), but there are no paths of length 3 or less (to connect $A$ and $K$ one must go through $H$—that requires at least two edges; to connect $H$ and $K$ requires at least two more edges.) It follows that the degree of separation between $A$ and $K$ is *four*.
- $N$ and $M$ are pretty far apart—the shortest possible path connecting them is $N, G, E, C, H, M$. There you are—five degrees of separation.
- There are several pairs of students in this seminar with six degrees of separation between them. Can you find a couple?

Social networks such as Facebook, Twitter, and LinkedIn are examples of *organic* networks—networks that evolve and change on their own without any organized or centralized planning. In contrast to these, there are *planned* networks that are designed with a specific purpose in mind. Our next example illustrates on a very small scale one of the most important planned networks in modern life—the electrical power grid.

## EXAMPLE 7.3 POWER GRIDS

Figure 7-2(a) shows a map of the main Texas power grid. Figure 7-2(b) shows a small section of the power grid connecting 14 small towns labeled $A$ through $N$. In this graph a small section of the vertices represent the towns and the edges represent the main power lines that carry electricity to the various towns. The graph is connected, so it is indeed a network—any town can draw power from the grid. In addition, there is a weight associated with each edge of the network. In this case the weight of an edge represents the length (in miles) of the power line connecting the two towns. [Figure 7-2(b) is not drawn to exact scale but is close. The reason for this is that the length of the power line is usually close to, but not necessarily the same, as the distance between the towns.]

We will return to this example in Section 7.3.




(a) (b)

FIGURE 7-2 (a) The main Texas power grid map. (b) Small section of the power grid.

The network in Fig. 7.2 is an example of what we call a **weighted network**. Each edge has a *weight* that we will generally think of as representing a *cost* (either money, time, or distance). We will deal with weighted networks later in the chapter. In almost all weighted network problems the weights are positive numbers, but later in this chapter we will make an exception and consider weighted networks where the weights are negative numbers.

## Trees

The second important concept in this chapter is that of a *tree*. We all know that trees are important, as they provide shade and help clean up our air, but in the context of this chapter the term **tree** means *a network that has no circuits.*

### EXAMPLE 7.4 TREES AND ROOTS

Figure 7-3 shows three networks.

- The network shown in Fig. 7-3(a) has the circuit $A, B, G, F, E, D, H, A$. It *is not* a tree.
- The network in Fig. 7-3(b) has no circuits. It *is* a tree. (What might look like a circuit in the picture is not—the crossing points of edge $HD$ with edges $BG$ and $CG$ are just crossing points and not vertices.) This tree may not look very tree-like, but we can fix that easily.
- It is obvious that the network in Fig. 7-3(c) has no circuits. It is a tree, and it looks the part. Surprisingly, this tree is the same tree as the one shown in Fig. 7-3(b). We just picked one of the vertices to be the "root" of the tree (in this case $B$) and built the "branches" of the tree up from the root. We can do this with any tree: Pick any vertex to be the root and build the tree up from there. Figure 7-4 shows two more versions of the same tree—in Fig. 7-4(a) the tree is rooted at $A$; in Fig. 7-4(b) the tree is rooted at $D$. (Both trees are shown sideways with the root on the left—just trying to not waste space on the page . . . and save some real trees!)

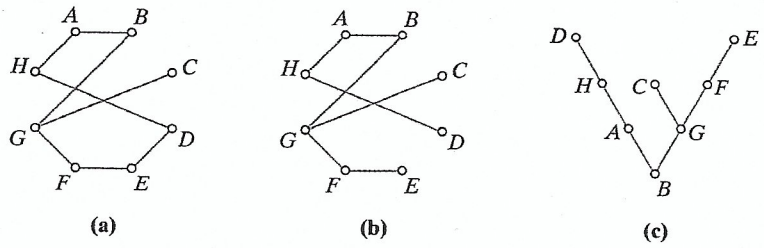



FIGURE 7-3 (a) A network with circuits is not a tree. (b) A tree. (c) The same tree with $B$ as the "root."

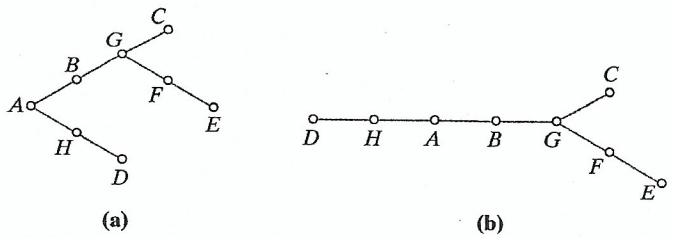



FIGURE 7-4 (a) Tree rooted at $A$. (b) Same tree rooted at $D$.

Trees have three key properties that distinguish them from ordinary networks. We will introduce the three properties first, illustrate them with an example, and conclude this section with a more formal version of these properties.

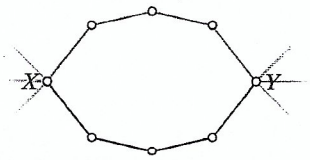



FIGURE 7-5 Two different paths joining $X$ and $Y$ make a circuit.

- **The single-path property.** In a tree, there is *only one path connecting two vertices.* If there were two paths connecting a pair of vertices, those two paths would create a circuit, as illustrated in Fig. 7-5. Conversely, a network that is not a tree must have at least one circuit, and that circuit will always provide alternative paths between its vertices. Look at Fig. 7-5 again: Given a circuit and two vertices ($X$ and $Y$) in the circuit, there are at least two different paths (red and blue) connecting $X$ and $Y$.

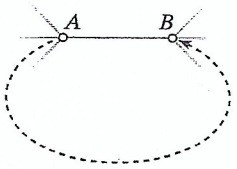



FIGURE 7-6 If $AB$ is not a bridge, then it must be part of a circuit.

- **The all-bridges property.** In a tree, *every edge is a bridge.* Essentially this means that a tree has no edges to spare—if we were to delete *any* edge, the tree would become disconnected and would no longer be a network. Figure 7-6 illustrates why every edge must be a bridge: Imagine an edge $AB$ that is *not* a bridge. Then there would have to be an alternate path from $A$ to $B$ (shown as the dashed red curve). But $AB$ together with the alternate path from $A$ to $B$ would form a circuit, and a tree doesn't have circuits. Conversely, if every edge of the network is a bridge then the network must be a tree.
- **The $N-1$ edges property.** A tree with $N$ vertices has $N-1$ edges. Always. This means that no matter what the shape of the tree is, the number of edges is one less than the number of vertices. The tree in Fig. 7-4 has $N=8$ vertices. We don't even have to check—the number of edges must be 7. Conversely, a network with $N$ vertices and $N-1$ edges must be a tree.

From the above properties of trees we inherit the following key property of networks: In a network with $N$ vertices and $M$ edges, $M \geq N-1$ (i.e., the number of edges is at least $N-1$). When $M=N-1$ the network is a tree; when $M>N-1$ the network has circuits. The difference between the number of edges $M$ and the minimum possible number of edges $N-1$ is an important number called the **redundancy** of the network.

- **Redundancy of a Network.** In a network with $N$ vertices and $M$ edges, the redundancy $R$ is given by $R=M-(N-1)$. [$R=0$ means the network is a tree; $R>0$ means the network is not a tree.]

## EXAMPLE 7.5 CONNECT THE DOTS (AND THEN STOP)

Imagine the following "connect-the-dots" game: Start with eight isolated vertices. The object of the game is to create a network connecting the vertices by adding edges, one at a time. You are free to create any network you want. In this game, bridges are good and circuits are bad. (Imagine, for example, that for each bridge in your network you get a \$10 reward, but for each circuit in your network you pay a \$10 penalty.)

So grab a marker and start playing. We will let $M$ denote the number of edges you have added at any point in time. In the early stages of the game ($M=1, 2, \ldots, 6$) the graph is disconnected [Figs. 7-7(a) through (d)].

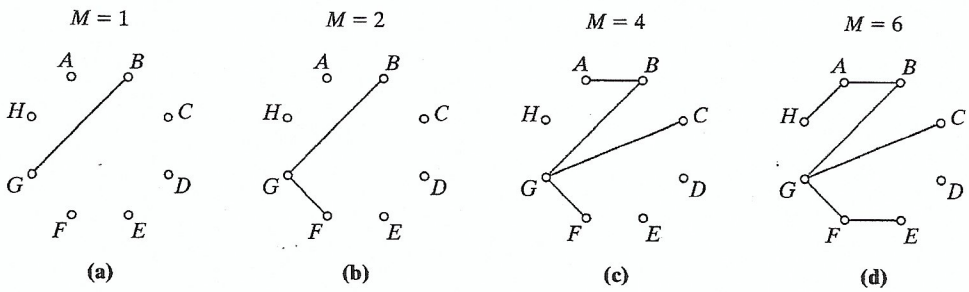



FIGURE 7-7 For small values of $M$, the graph is disconnected.

As soon as you get to $M=7$ (and if you stayed away from forming any circuits) the graph becomes connected. Some of the possible configurations are shown in Fig. 7-8. Each of these networks has redundancy $R=0$ and is, therefore, a tree, and now each of the seven edges is a bridge. Stop here and you will come out \$70 richer.

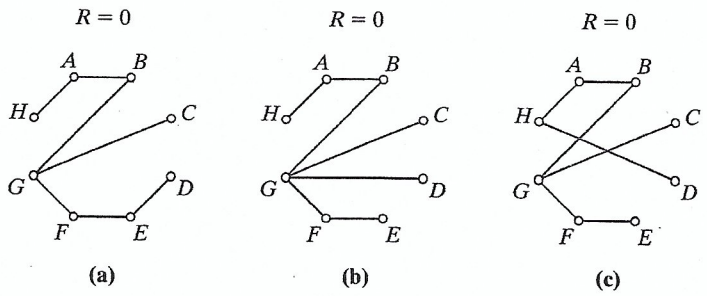



FIGURE 7-8 When $M = 7$, we have a tree ($R = 0$).

Interestingly, this is as good as it will get. When $M = 8$ ($R = 1$), the graph will have a circuit—it just can't be avoided. In addition, none of the edges in that circuit can be bridges of the graph. As a consequence, the larger the circuit that we create, the fewer the bridges left in the graph. [The graph in Fig. 7-9(a) has a circuit and five bridges, the graph in Fig. 7-9(b) has a circuit and two bridges, and the graph in Fig. 7-9(c) has a circuit and only one bridge.] As the redundancy increases, the number of circuits goes up (very quickly) and the number of bridges goes down [Figs. 7-10(a), (b), and (c)].

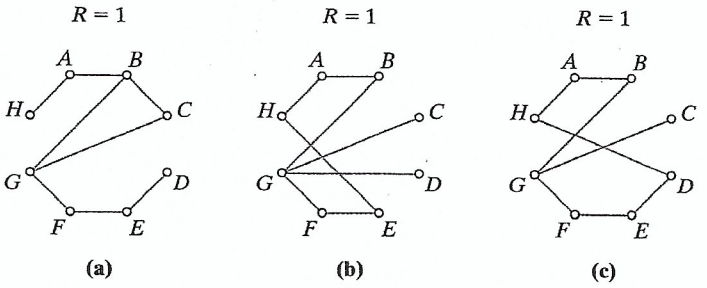



FIGURE 7-9 When $M = 8$ ($R = 1$) we have a network with a circuit.

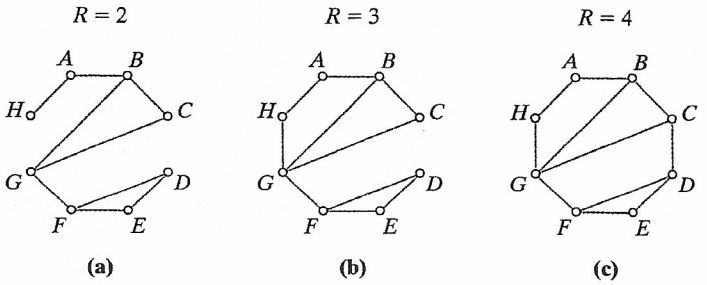



FIGURE 7-10 For larger values of $R$, we have a network with lots of circuits.

Our preceding discussion of trees and networks can be rephrased and summarized in the following two key observations (and respective conclusions):

- A tree is a *minimally connected* network. This means that every edge of the tree is needed to keep it connected—in a tree every edge is a bridge, and there are no redundant edges (zero redundancy). This also means that the number of edges is always one less than the number of vertices ($R = 0$ implies $M = N - 1$). *Conclusion 1: If you want to minimize the number of edges in a network, build a tree.*
- A network that is not a tree must have some redundant edges (positive redundancy). The redundant edges form circuits, and the higher the redundancy the more circuits in the network. Each circuit creates additional paths for connecting vertices in the circuit, so the more circuits the more ways there are to get around in the network. *Conclusion 2: If you want to have lots of alternative routes to get around in a network, increase its redundancy.*

# 7.2 Spanning Trees, MSTs, and MaxSTs

## Spanning Trees

A **spanning tree** in a network is a *subtree* of the network that *spans* all the vertices. The easiest way to explain the meaning of this definition is with a few examples.

### EXAMPLE 7.6 SUBTREES AND SPANNING TREES

Figure 7-11(a) shows a small network with 8 vertices and 9 edges. Figure 7-11(b) shows (in red) a *subtree* of the network. The name *subtree* comes from the fact that the red tree has its vertices and edges inside the network. The subtree in Fig. 7-11(b) does not include all the vertices, but the one in Fig. 7-11(c) does. We say that the subtree in Fig. 7-11(c) *spans* the network, and we call such subtrees *spanning trees* of the network. The spanning tree in Fig. 7-11(c) has 7 edges, and any other spanning tree of this network will have 7 edges as well.

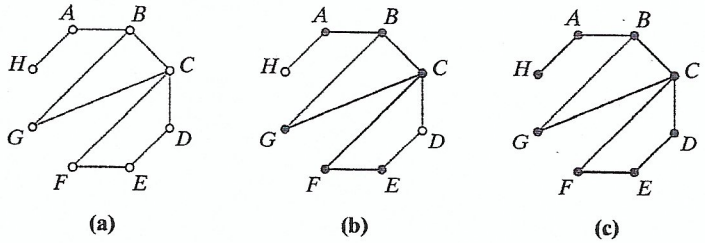



FIGURE 7-11 (a) The original network. (b) A subtree (in red). Vertices *H* and *D* are not in the subtree. (c) A spanning subtree.

### EXAMPLE 7.7 COUNTING SPANNING TREES

The network in Fig. 7-12(a) has $N = 8$ vertices and $M = 8$ edges. The redundancy of the network is $R = 1$. To find a spanning tree we will have to "discard" one edge. Five of these edges are bridges of the network, and they *will have to be part of any spanning tree.* The other three edges ($BC$, $CG$, and $GB$) form a circuit of length 3, and if we exclude any one of the three edges, then we will have a spanning tree. Thus, the network has three different spanning trees [Figs. 7-12(b), (c), and (d)].

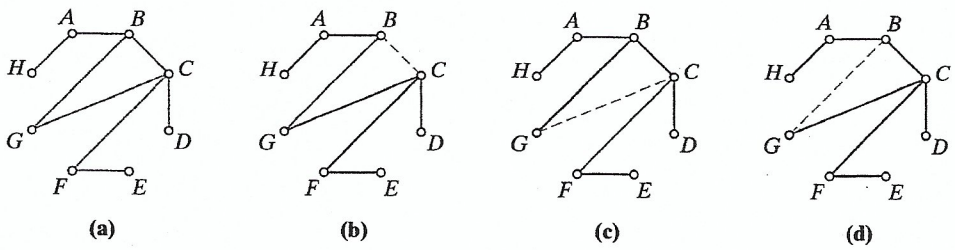



FIGURE 7-12 (a) The original network. (b), (c), and (d) Spanning trees.

The network in Fig. 7-13(a) is the same network as the one in Fig. 7-11(a). The redundancy of the network is $R = 2$, so to find a spanning tree we will have to "discard" two edges. Edges $AB$ and $AH$ are bridges of the network, so they will have to be part of any spanning tree. The other seven edges are split into two separate circuits ($B$, $C$, $G$, $B$ of length 3 and $C$, $D$, $E$, $F$, $C$ of length 4). A spanning tree can be found by "busting" each of the two circuits. This means excluding any one of the three edges of circuit $B$, $C$, $G$, $B$ and any one of the four edges of circuit

$C, D, E, F, C$. For example, if we exclude $BC$ and $CD$, we get the spanning tree shown in Fig. 7-13(b). We could also exclude $BC$ and $DE$ and get the spanning tree shown in Fig. 7-13(c), and so on. Given that there are $3 \times 4 = 12$ different ways to choose an edge from the circuit of length 3 and an edge from the circuit of length 4, we will not show all 12 spanning trees. [Figs. 7-13(b) through (e) show some of them.]

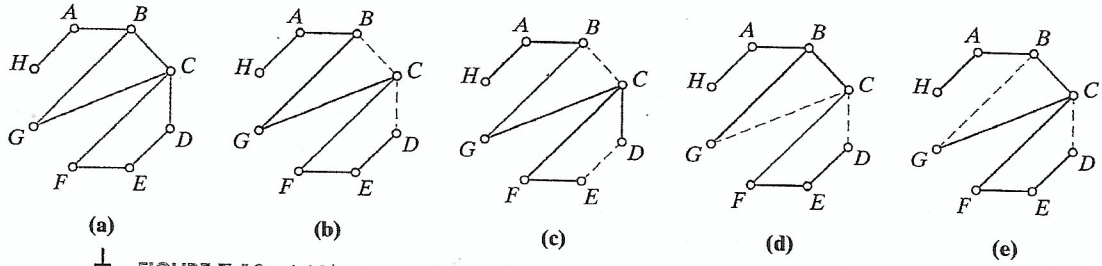



FIGURE 7-13 (a) The original network. (b), (c), (d), and (e) Spanning trees.

## EXAMPLE 7.8 MORE COUNTING OF SPANNING TREES

The network in Fig. 7-14(a) is another network with 8 vertices and redundancy $R = 2$. The difference between this network and the one in Fig. 7-13(a) is that here the circuits $B, C, G, B$ and $C, D, E, G, C$ share a common edge $CG$. Determining which pairs of edges can be excluded in this case is a bit more complicated.

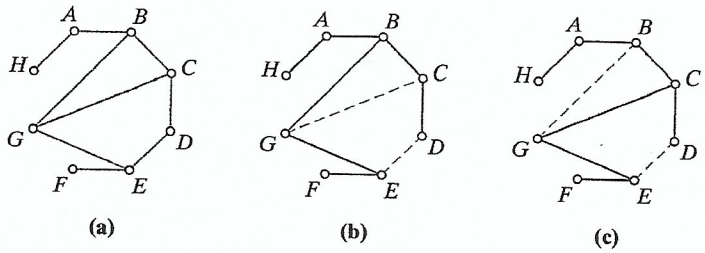



FIGURE 7-14 (a) The original network. (b) and (c) Spanning trees.

If one of the excluded edges is the common edge $CG$, then the other excluded edge can be any other edge in a circuit. There are five choices ($BC$, $CD$, $DE$, $EG$, and $GB$), and each choice will result in a different spanning tree [one of these is shown in Fig. 7-14(b)]. The alternative scenario is to exclude two edges neither of which is the common edge $CG$. In this case one excluded edge has to be either $BC$ or $BG$ (to "bust" circuit $B, C, G, B$), and the other excluded edge has to be either $CD$, $DE$, or $EG$ (to "bust" circuit $C, D, E, G, C$). There are $2 \times 3 = 6$ possible spanning trees that can be formed this way [one of these is shown in Fig. 7-14(c)]. Combining the two scenarios gives a total of 11 possible spanning trees for the network in Fig. 7-14(a).

As the redundancy of a network grows, the number of spanning trees gets very large. In our next couple of examples we consider spanning trees in weighted networks of high redundancy.

## EXAMPLE 7.9 THE AMAZONIAN CABLE NETWORK

The Amazonia Telephone Company is contracted to provide telephone, cable, and Internet service to the seven small mining towns shown in Fig. 7-15(a). These towns are located deep in the heart of the Amazon jungle, which makes the project particularly difficult and expensive. In this environment the most practical and environmentally friendly option is to create a network of underground fiber-optic cable lines connecting the towns. In addition, it makes sense to bury the underground

cable lines along the already existing roads connecting the towns. The existing network of roads is shown in Fig. 7-15(a). Figure 7-15(b) is a network model of all the possible connections between the towns. The vertices of the network represent the towns, the edges represent the existing roads, and the weight of each edge represents the cost (in millions of dollars) of creating a fiber-optic cable connection along that particular edge.

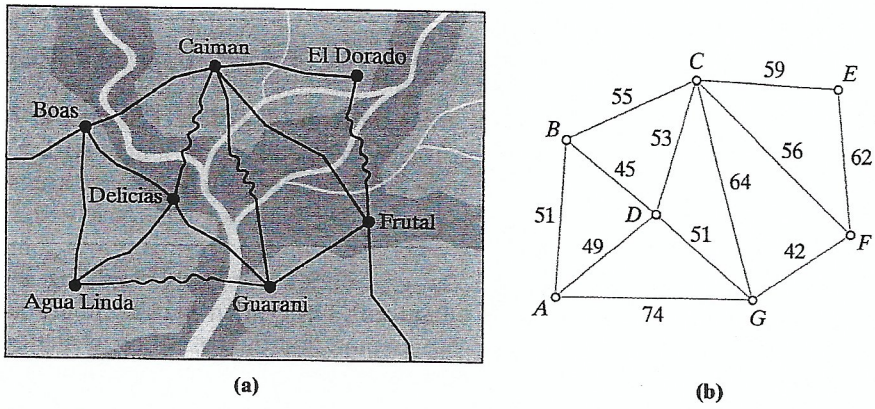



FIGURE 7-15 (a) Network of roads connecting seven towns. (b) Network model showing the cost (in millions) of each connection.

The problem facing the engineers and planners of the Amazonia Telephone Company is how to design a cable network that connects all the towns using the existing network of roads and that costs as little as possible to build—in other words, a *minimum cost* network. The first thing we can say about a minimum cost network is that it must be a *spanning tree* of the original network of roads shown in Fig. 7-15(a). (To connect all the towns means that the cable network must *span* all the vertices; to eliminate redundant connections means that it will have redundancy $R = 0$, and that means a *tree*.) But in this example there are costs involved, and not all spanning trees will have the same cost, so rather than finding any old spanning tree, we need to find the spanning tree with least total weight. We call such a spanning tree a *minimum spanning tree*.

One way to find a minimum spanning tree is to list all possible spanning trees, find the total cost of each, and pick the one with least cost. (This is the same approach we described in Chapter 6 as the *brute-force algorithm*.) The problem is that the network in Fig. 7-15(b) has high redundancy ($R = 6$) and hundreds of possible spanning trees. Sifting through all of them to find the one with least cost is not a good plan. In Section 7.3 we will discuss a better way of finding a minimum spanning tree in the Amazon jungle.

Example 7.9 was our introduction to the concept of a *minimum spanning tree*. We now give it a formal definition:

- Minimum Spanning Tree. In a weighted network, a **minimum spanning tree (MST)** is *a* spanning tree with least total weight.

Sometimes there is only one MST and we can refer to it as *the* MST, but we can't assume this to be true in general. (For example, if all the weights in the network are the same, then every spanning tree is an MST.)

In most applications, the weights of the network represent *costs*—money, time, or distance. In these cases the goal is to *minimize*. There are some applications, however, where the weights represent *profits*. Profits (or other types of *gains* such as *higher bandwidth* on Internet connections or *increased flows* in pipelines) are things

we want more of, rather than less, so instead of minimizing we should be maximizing. This leads to our next definition.

- Maximum Spanning Tree. In a weighted network, a **maximum spanning tree (MaxST)** is *a* spanning tree with highest total weight.

Our next example illustrates a MaxST application.

**EXAMPLE 7.10** THE AMAZON MAX PROFIT NETWORK

This example is the flip side of Example 7.9. The problem is still to connect the seven towns in the Amazon jungle with a cable network, but the circumstances are quite different. Imagine that there are two parties involved: one party—say the government—is paying for the construction costs of the cable network; the other party—say the telephone company—is going to operate and run the network.

The government insists on building a network with zero redundancy (i.e., a spanning tree). Other than that, there are no restrictions on the choice of spanning tree (we assume that the cost of construction is the same no matter which spanning tree gets built, so to the paying party any spanning tree will do). On the other hand, to the telephone company the choice of spanning tree is very important—as is the case with any company, it wants to maximize profit, so building the most profitable spanning tree is the name of the game.

Let's assume that the weight of each edge of the network in Fig. 7-16 represents the expected annual *profits* (in millions) to the phone company for operating that segment of the network. (You may have noticed that we are using exactly the same weighted network as that in Example 7.9. As you will see in the next section, this is not a coincidence.) To the phone company, the problem now becomes finding the MaxST of the original network. We will learn how to do this in the next section.

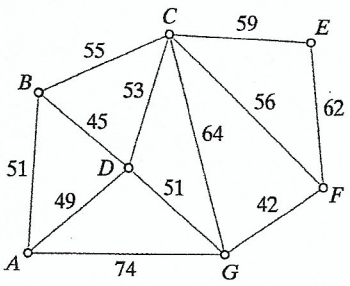


FIGURE 7-16

## 7.3 Kruskal's Algorithm

Unlike the situation with TSPs (see Chapter 6) there are several *efficient* and *optimal* algorithms for finding minimum spanning trees. Moreover, any algorithm that finds minimum spanning trees can be tweaked to find maximum spanning trees as well. In this section we will introduce one such algorithm—a simple algorithm called *Kruskal's algorithm* after American mathematician Joseph Kruskal.

Kruskal's algorithm is very similar to the *cheapest-link algorithm* used for solving TSPs in Chapter 6. The minimum spanning tree gets built one edge at a time by choosing at each step the cheapest available edge. The only restriction in choosing the edges is that one should never choose an edge that creates a circuit. Having three or more edges coming out of a vertex, however, is now OK. Continue choosing edges this way until $N-1$ edges are chosen. At that point one has an MST.

The following is a formal description of Kruskal's algorithm. (For simplicity, we use "cheapest" to denote "of least weight.")

**KRUSKAL'S ALGORITHM**

- **Step 1.** Pick the *cheapest edge* available. (In case of a tie, pick one at random.) Mark it (say in red).
- **Step 2.** Pick the next cheapest edge available and mark it.
- **Steps 3, 4, . . . , $N-1$.** Continue picking and marking the cheapest unmarked edge available that does not create a circuit. After step $N-1$ you are done.

**EXAMPLE 7.11** THE AMAZONIAN CABLE NETWORK AND KRUSKAL'S ALGORITHM

In Example 7.9 we raised the following question: What is the optimal fiber-optic cable network connecting the seven towns shown in Fig. 7-17(a)? The weight of each edge is the cost (in millions of dollars) of laying the cable along that segment of the network.

The answer, as we now know, is to find the minimum spanning tree of the network in Fig. 7.17(a). We will use Kruskal's algorithm to do it. Here are the details:

- Step 1. We start by choosing the cheapest edge of the network. In this case we choose *GF*, and mark it in red (or any other color) as shown in Fig. 7-17(b). (Note that this does not have to be the first link actually built—we are putting the network together on paper only. On the ground, the schedule of construction is a different story, and there are many other factors that need to be considered.)

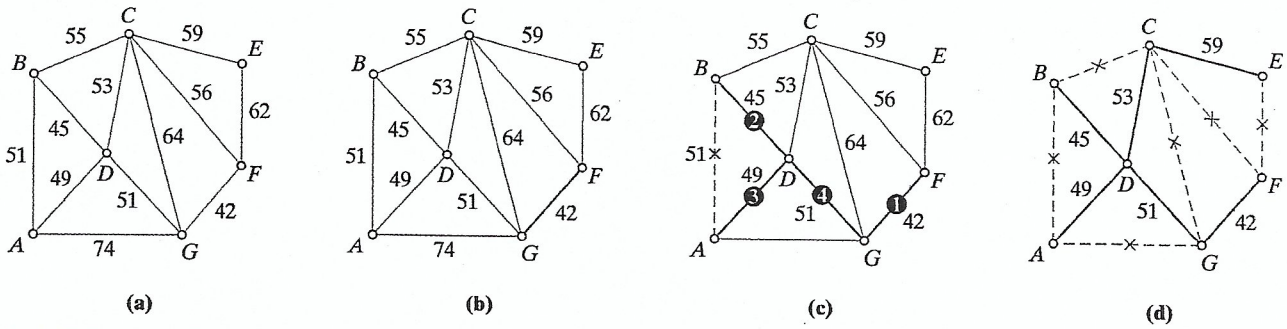



FIGURE 7-17 (a) The original network. (b) Start. (c) First four steps. (d) The MST.

- Step 2. The next cheapest edge available is *BD* at \$45 million. We choose it for the MST and mark it in red.
- Step 3. The next cheapest edge available is *AD* at \$49 million. Again, we choose it for the MST and mark it in red.
- Step 4. For the next cheapest edge there is a tie between *AB* and *DG*, both at \$51 million. But we can rule out *AB*—it would create a circuit in the MST, and we can't have that! (For bookkeeping purposes it is a good idea to erase or cross out the edge.) The edge *DG*, on the other hand, is just fine, so we mark it in red and make it part of the MST.

  Figure 7-17(c) shows how things look at this point.
- Step 5. The next cheapest edge available is *CD* at \$53 million. No problems here, so again, we mark it in red and make it part of the MST.
- Step 6. The next cheapest edge available is *BC* at \$55 million, but this edge would create a circuit, so we cross it out. The next possible choice is *CF* at \$56 million, but once again, this choice creates a circuit so we must cross it out. The next possible choice is *CE* at \$59 million, and this is one we do choose. We mark it in red and make it part of the MST.
- Step. . . . Wait a second—we are finished! Even without looking at a picture, we can tell we are done—six links is exactly what is needed for an MST on seven vertices.

Figure 7-17(d) shows the MST in red. The total cost of the network is \$299 million.

Any algorithm that can find an MST can also be used to find a MaxST by means of a simple modification: *Change the signs of the weights* in the network. We call the network that we get when we change the signs of all the weights the **negative** of the

original network. Switching the signs of the weights switches MaxSTs into MSTs and vice versa. *A MaxST of a network is an MST of the negative network.* This follows from the simple fact that changing the signs of numbers reverses their inequality relationships ($74 > 45$ implies that $-74 < -45$). We will illustrate this idea by returning to the MaxST problem introduced in Example 7.10.

## EXAMPLE 7.12 THE AMAZON MAX PROFIT AND KRUSKAL'S ALGORITHM

Figure 7-18(a) shows the *negative* network for the original network in Example 7.10 (Fig. 7-16). We'll use Kruskal's algorithm to find the MST of this negative network. This will be the MaxST that we are looking for.

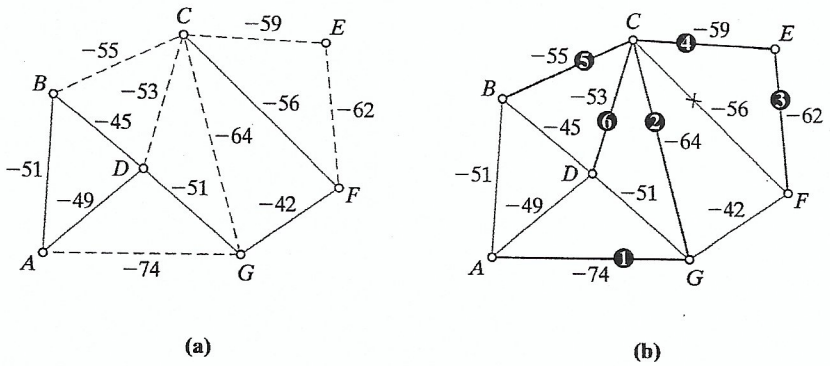



FIGURE 7-18 (a) The *negative* network. (b) The MST of the negative network.

- Step 1. The cheapest edge in the negative network is $AG$, with a weight of $-74$. We choose it and mark it in red.
- Step 2. The next cheapest edge is $CG$, with a weight of $-64$. We choose it and mark it in red.
- Step 3. The next cheapest edge is $EF$, with a weight of $-62$. We choose it and mark it in red.
- Step 4. The next cheapest edge is $CE$, with a weight of $-59$. We choose it and mark it in red.
- Step 5. The next cheapest edge is $CF$, with a weight of $-56$. We choose it and . . . oops! Can't do that. Choosing $CF$ would create a red circuit. Spanning trees don't have circuits. We rule out $CF$, and move on to the next cheapest edge, $CB$, with a weight of $-55$. No circuits with this one, so we choose it for the MST.
- Step 6. The next cheapest edge is $CD$, with a weight of $-53$. No problems here, so we choose it and mark it in red.

At Step 6 we are done! We have our MST for the negative network, shown in Fig. 7-18(b). This network is the MaxST we were looking for in the original network. The total profits that can be expected from this MaxST are given by the sum of the original weights: $74 + 64 + 62 + 59 + 55 + 53 = 367$ million. ■

## EXAMPLE 7.13 FINDING MSTs IN POWER GRIDS USING KRUSKAL'S ALGORITHM

In many rural areas electricity transmission lines are old. Old transmission lines carry lower voltages, leak more power, and are more sensitive to bad weather than modern transmission lines, so a common infrastructure project is to update the older parts of the grid with new transmission lines. The problem is how to choose the newer transmission lines so that they carry power to as many customers as possible

while at the same time keeping the cost of the project down. The least costly solution often involves finding an MST.

Figure 7-19(a) shows a section of the electrical power grid connecting 14 rural towns in central Texas. This is the network introduced in Example 7.3. The weight of each edge represents the length (in miles) of that particular segment of the grid. When the terrain is flat—as is in central Texas—the cost of replacing a transmission line is proportional to the length of the line (for high-voltage, modern transmission lines it is about $500,000 per mile), so when we minimize length we are also minimizing cost. It follows that the MST of the grid in Fig. 7-19(a) is going to give the cheapest spanning tree of updated power lines for the grid.

We'll find the MST using Kruskal's algorithm. [As a heads-up, we know ahead of time that in this network Kruskal's algorithm will require 13 steps (the network has 14 vertices), so we'll be brief and to the point.]

- Step 1. Choose *LM* (5) and mark it in red.
- Step 2. Choose *AJ* (11) and mark it in red.
- Step 3. Choose *EF* (11) and mark it in red.
- Step 4. Choose *FG* (12) and mark it in red.
- Step 5. Choose *HM* (13) and mark it in red.
- Step 6. Choose *NK* (14) and mark it in red.
- Step 7. Choose *BK* (16) and mark it in red.
- Step 8. Choose *HN* (18) and mark it in red.
- Step 9. Skip *NL* (21) because it closes a circuit. Choose *AB* (23) and mark it in red.
- Step 10. Choose *DE* (23) and mark it in red.
- Step 11. Choose *CD* (26) and mark it in red.
- Step 12. Choose *FM* (30) and mark it in red.
- Step 13. Skip *CE* (31), *BC* (31), and *JK* (32) because they all close circuits. Choose *IN* (32) and mark it in red. That's it—we are done.

Figure 7-19(b) shows the MST. The total length of the MST is 232 miles. At an average cost of $500,000 per mile, the total cost of the infrastructure update is $117 million.

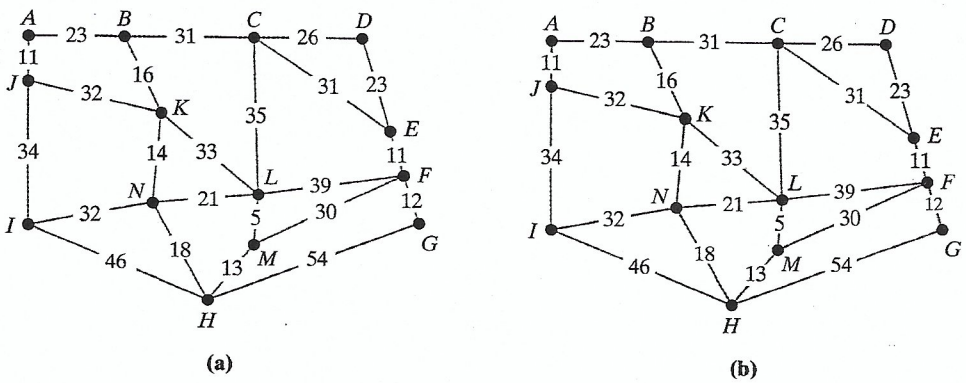



FIGURE 7-19 (a) The power grid. (b) The MST for infrastructure update.

# EXERCISES

## WALKING

### 7.1 Networks and Trees

1. A computer lab has seven computers labeled $A$ through $G$. The connections between computers are as follows:
   - $A$ is connected to $D$ and $G$
   - $B$ is connected to $C$, $E$, and $F$
   - $C$ is connected to $B$, $E$, and $F$
   - $D$ is connected to $A$ and $G$
   - $E$ is connected to $B$ and $C$
   - $F$ is connected to $B$ and $C$
   - $G$ is connected to $A$ and $D$

   Is the lab set-up a computer network? Explain why or why not.

2. The following is a list of the electrical power lines connecting eight small towns labeled $A$ through $H$.
   - A power line connecting $A$ and $D$
   - A power line connecting $B$ and $C$
   - A power line connecting $B$ and $E$
   - A power line connecting $B$ and $G$
   - A power line connecting $C$ and $G$
   - A power line connecting $D$ and $F$
   - A power line connecting $D$ and $H$
   - A power line connecting $E$ and $G$

   Do the power lines form a network? Explain why or why not.

3. Consider the network shown in Fig. 7-20.
   (a) How many degrees of separation are there between $C$ and $E$?
   (b) How many degrees of separation are there between $A$ and $E$?
   (c) How many degrees of separation are there between $A$ and $H$?

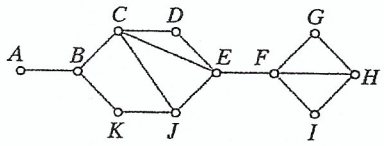


FIGURE 7-20

4. Consider the network shown in Fig. 7-21.
   (a) How many degrees of separation are there between $D$ and $J$?
   (b) How many degrees of separation are there between $A$ and $L$?
   (c) How many degrees of separation are there between $A$ and $K$?

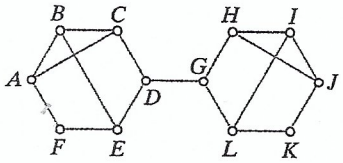


FIGURE 7-21

5. Consider the tree shown in Fig. 7-22 on the next page.
   (a) How many degrees of separation are there between $A$ and $J$?
   (b) How many degrees of separation are there between $E$ and $L$?
   (c) How many degrees of separation are there between $M$ and $P$?
   (d) What is the largest degree of separation between a pair of vertices?

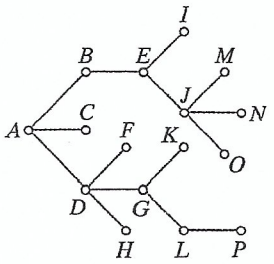



FIGURE 7-22

**6.** Consider the tree shown in Fig. 7-23.

**(a)** How many degrees of separation are there between $A$ and $P$?

**(b)** How many degrees of separation are there between $E$ and $P$?

**(c)** How many degrees of separation are there between $L$ and $P$?

**(d)** What is the largest degree of separation between a pair of vertices?

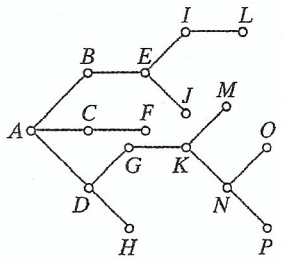



FIGURE 7-23

*In Exercises 7 through 20 you are given information about a network. Choose one of the following three options: (A) the network is definitely a tree; (B) the network is definitely not a tree; (C) the network may or may not be a tree (more information is needed). Accompany your answer with a brief explanation for your choice.*

**7.** The network has 15 vertices and 16 edges.

**8.** The network has 23 vertices and no bridges.

**9.** The network has 16 vertices and 15 edges.

**10.** The network has 23 vertices and 22 bridges.

**11.** The network has redundancy $R = 1$.

**12.** The network has redundancy $R = 0$.

**13.** The network has 10 vertices ($A$ through $J$), and there is only one path connecting $A$ and $J$.

**14.** The network has 10 vertices ($A$ through $J$) and there are two paths connecting $C$ and $D$.

**15.** The network has five vertices, no loops, and no multiple edges, and every vertex has degree 4.

**16.** The network has five vertices, no loops, and no multiple edges, and every vertex has degree 2.

**17.** The network has five vertices, no loops, and no multiple edges, and has one vertex of degree 4 and four vertices of degree 1.

**18.** The network has five vertices, no loops, and no multiple edges, and has two vertices of degree 1 and three vertices of degree 2.

**19.** The network has all vertices of even degree. (*Hint*: You will need to use some concepts from Chapter 5 to answer this question.)

**20.** The network has two vertices of odd degree and all the other vertices of even degree. (*Hint*: You will need to use some concepts from Chapter 5 to answer this question.)

## 7.2 Spanning Trees, MSTs, and MaxSTs

**21.** Consider the network shown in Fig. 7-24.

**(a)** Find a spanning tree of the network.

**(b)** Calculate the redundancy of the network.

**(c)** What is the largest degree of separation between a pair of vertices in the network?

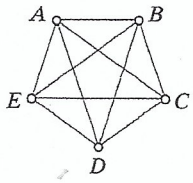



FIGURE 7-24

**22.** Consider the network shown in Fig. 7-25.

**(a)** Find a spanning tree of the network.

**(b)** Calculate the redundancy of the network.

**(c)** What is the largest degree of separation between a pair of vertices in the network?

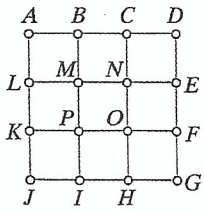



FIGURE 7-25

**23.** Consider the network shown in Fig. 7-26.

**(a)** Find a spanning tree of the network.

**(b)** Calculate the redundancy of the network.

**(c)** What is the largest degree of separation between a pair of vertices in the network?

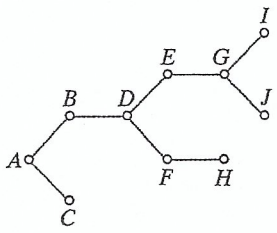



FIGURE 7-26

**24.** Consider the network shown in Fig. 7-27.

**(a)** Find a spanning tree of the network.

**(b)** Calculate the redundancy of the network.

**(c)** What is the largest degree of separation between a pair of vertices in the network?

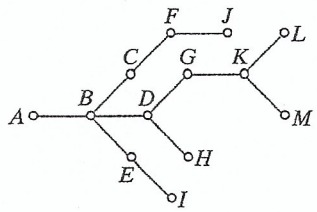



FIGURE 7-27

**25.** **(a)** Find all the spanning trees of the network shown in Fig. 7-28(a).

**(b)** Find all the spanning trees of the network shown in Fig. 7-28(b).

**(c)** How many different spanning trees does the network shown in Fig. 7-28(c) have?

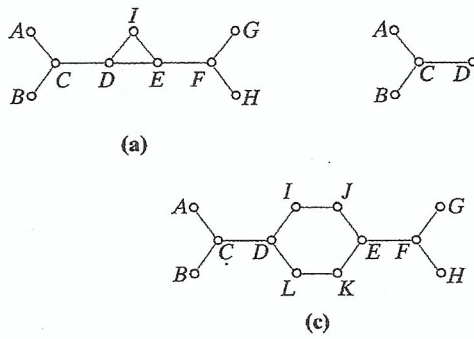



FIGURE 7-28

**26.** **(a)** Find all the spanning trees of the network shown in Fig. 7-29(a).

**(b)** Find all the spanning trees of the network shown in Fig. 7-29(b).

**(c)** How many different spanning trees does the network shown in Fig. 7-29(c) have?

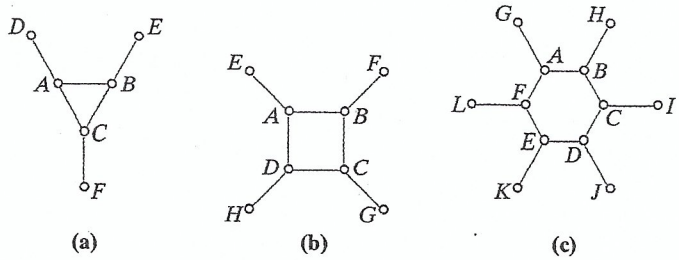



FIGURE 7-29

**27.** **(a)** How many different spanning trees does the network shown in Fig. 7-30(a) have?

**(b)** How many different spanning trees does the network shown in Fig. 7-30(b) have?

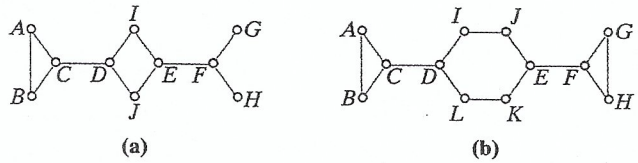



FIGURE 7-30

**28.** **(a)** How many different spanning trees does the network shown in Fig. 7-31(a) have?

**(b)** How many different spanning trees does the network shown in Fig. 7-31(b) have?

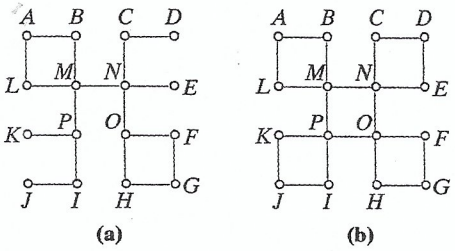



FIGURE 7-31

**29.** Consider the network shown in Fig. 7-32.

**(a)** How many different spanning trees does this network have?

**(b)** Find the spanning tree that has the largest degree of separation between $H$ and $G$.

**(c)** Find a spanning tree that has the smallest degree of separation between $H$ and $G$.

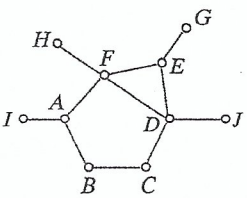



FIGURE 7-32

**30.** Consider the network shown in Fig. 7-33.

(a) How many different spanning trees does this network have?

(b) Find a spanning tree that has the largest degree of separation between $H$ and $J$.

(c) Find a spanning tree that has the smallest degree of separation between $K$ and $G$.

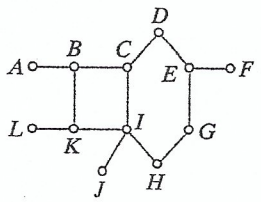



FIGURE 7-33

## 7.3 Kruskal's Algorithm

**31.** The 3 by 4 grid shown in Fig. 7-34 represents a network of streets (3 blocks by 4 blocks) in a small subdivision. For landscaping purposes, it is necessary to get water to each of the corners by laying down a system of pipes along the streets. The cost of laying down the pipes is \$40,000 per mile, and each block of the grid is exactly half a mile long. Find the cost of the cheapest network of pipes connecting all the corners of the subdivision. Explain your answer. (*Hint*: First determine the number of blocks in the MST.)

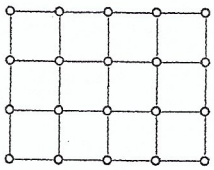


FIGURE 7-34

**32.** The 4 by 5 grid shown in Fig. 7-35 represents a network of streets (4 blocks by 5 blocks) in a small subdivision. For landscaping purposes, it is necessary to get water to each of the corners by laying down a system of pipes along the streets. The cost of laying down the pipes is \$40,000 per mile, and each block of the grid is exactly half a mile long. Find the cost of the cheapest network of pipes connecting all the corners of the subdivision. Explain your answer. (*Hint*: First determine the number of blocks in the MST.)

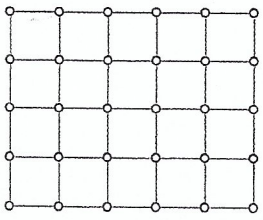


FIGURE 7-35

**33.** Find the MST of the network shown in Fig. 7-36 using Kruskal's algorithm, and give its weight.

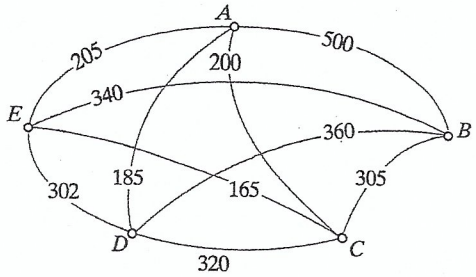



FIGURE 7-36

**34.** Find the MST of the network shown in Fig. 7-37 using Kruskal's algorithm, and give its weight.

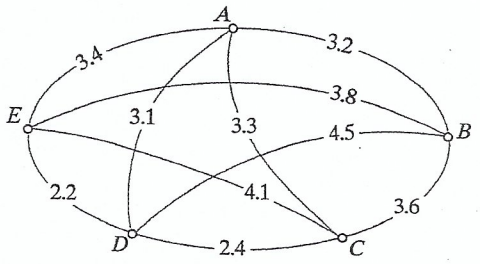



FIGURE 7-37

**35.** Find the MST of the network shown in Fig. 7-38 using Kruskal's algorithm, and give its weight.

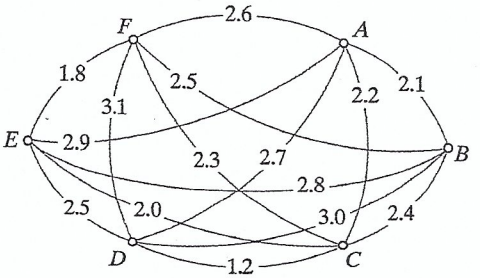



FIGURE 7-38

**36.** Find the MST of the network shown in Fig. 7-39 using Kruskal's algorithm, and give its weight.

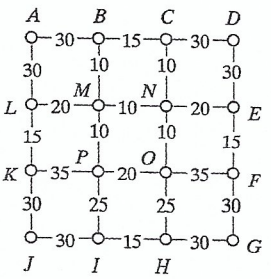



FIGURE 7-39

**37.** Find the MaxST of the network shown in Fig. 7-36 using Kruskal's algorithm and give its weight.

**38.** Find the MaxST of the network shown in Fig. 7-37 using Kruskal's algorithm and give its weight.

**39.** Find the MaxST of the network shown in Fig. 7-38 using Kruskal's algorithm and give its weight.

**40.** Find the MaxST of the network shown in Fig. 7-39 using Kruskal's algorithm and give its weight.

## JOGGING

**41.** The mileage chart in Fig. 7-40 shows the distances between Atlanta, Columbus, Kansas City, Minneapolis, Pierre, and Tulsa. Working directly from the mileage chart use Kruskal's algorithm to find the MST connecting the six cities. (*Hint*: See Example 6.15 for the use of an auxiliary graph.)

**Mileage Chart**

| | Atlanta | Columbus | Kansas City | Minneapolis | Pierre | Tulsa |
|---|---|---|---|---|---|---|
| **Atlanta** | * | 533 | 798 | 1068 | 1361 | 772 |
| **Columbus** | 533 | * | 656 | 713 | 1071 | 802 |
| **Kansas City** | 798 | 656 | * | 447 | 592 | 248 |
| **Minneapolis** | 1068 | 713 | 447 | * | 394 | 695 |
| **Pierre** | 1361 | 1071 | 592 | 394 | * | 760 |
| **Tulsa** | 772 | 802 | 248 | 695 | 760 | * |

FIGURE 7-40

**42.** The mileage chart in Fig. 7-41 shows the distances between Boston, Dallas, Houston, Louisville, Nashville, Pittsburgh, and St. Louis. Working directly from the mileage chart use Kruskal's algorithm to find the MST connecting the seven cities. (*Hint*: See Example 6.15 for the use of an auxiliary graph.)

**Mileage Chart**

| | Boston | Dallas | Houston | Louisville | Nashville | Pittsburgh | St. Louis |
|---|---|---|---|---|---|---|---|
| **Boston** | * | 1748 | 1804 | 941 | 1088 | 561 | 1141 |
| **Dallas** | 1748 | * | 243 | 819 | 660 | 1204 | 630 |
| **Houston** | 1804 | 243 | * | 928 | 769 | 1313 | 779 |
| **Louisville** | 941 | 819 | 928 | * | 168 | 388 | 263 |
| **Nashville** | 1088 | 660 | 769 | 168 | * | 553 | 299 |
| **Pittsburgh** | 561 | 1204 | 1313 | 388 | 553 | * | 588 |
| **St. Louis** | 1141 | 630 | 779 | 263 | 299 | 588 | * |

FIGURE 7-41

**43.** Figure 7-42(a) shows a network of roads connecting cities $A$ through $G$. The weights of the edges represent the cost (in millions of dollars) of putting underground fiber-optic lines along the roads, and the MST of the network is shown in red. Figure 7-42(b) shows the same network except that one additional road (connecting $E$ and $G$) has been added. Let $x$ be the cost (in millions) of putting fiber-optic lines along this new road.

(a) Describe the MST of the network in Fig. 7-42(b) in the case $x > 59$. Explain your answer.

(b) Describe the MST of the network in Fig. 7-42(b) in the case $x < 59$. Explain your answer.

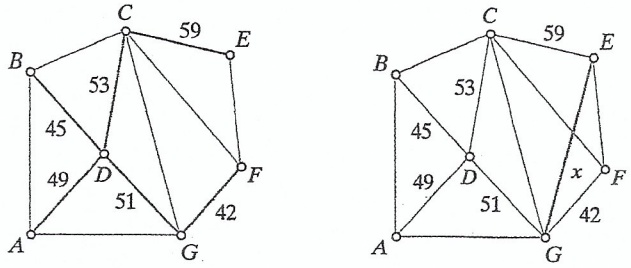



FIGURE 7-42

**44.** Consider a network with $M$ edges. Let $k$ denote the number of bridges in the network.

(a) If $M = 5$, list all the possible values of $k$.

(b) If $M = 123$, describe the set of all possible values of $k$.

**45.** (a) Let $G$ be a tree with $N$ vertices. Find the sum of the degrees of all the vertices in $G$.

(b) Explain why a tree must have at least two vertices of degree 1. (A vertex of degree 1 in a tree is called a *leaf*.)

(c) Explain why in a tree with three or more vertices the degrees of the vertices cannot all be the same.

**46.** Explain why in a network with no loops or multiple edges, the maximum redundancy is given by $R = \frac{(N^2 - 3N + 2)}{2}$. (*Hint*: The maximum redundancy occurs when the network is $K_N$.)

**47.** This exercise refers to weighted networks where all the weights in the network are different. Explain why these networks have only one MST and one MaxST. (*Hint*: Think of Kruskal's algorithm.)

**48.** This exercise refers to weighted networks where the weights in the network are *not* all different (i.e., there are at least two edges with the same weight).

(a) Give an example of a network of this type that has only one MST.

(b) Give an example of a network of this type that has more than one MST.

**49.** Suppose that in a weighted network there is just one edge (call it $XY$) with the *smallest* weight. Explain why the edge $XY$ must be in every MST of the network.

**50.** Suppose that in a weighted network there is just one edge (call it $XY$) with the *largest* weight.

**(a)** Give an example of a network with more than one MST and such that $XY$ must be in every MST.

**(b)** Give an example of a network with more than one MST and such that $XY$ is in none of the MSTs.

**51.** Suppose $G$ is a disconnected graph with $N$ vertices, $M$ edges, and no circuits.

**(a)** How many components does the graph have when $N = 9$ and $M = 6$?

**(b)** How many components does the graph have when $N = 240$ and $M = 236$? Explain your answer.

**52.** Suppose $G$ is a disconnected graph with no circuits. Let $N$ denote the number of vertices, $M$ the number of edges, and $K$ the number of components. Explain why $M = N - K$. (*Hint*: Try Exercise 51 first.)

**53. Cayley's theorem.** Cayley's theorem says that the number of spanning trees in a complete graph with $N$ vertices is given by $N^{N-2}$.

**(a)** List the $4^2 = 16$ spanning trees of $K_4$.

**(b)** Which is larger, the number of Hamilton circuits or the number of spanning trees in a complete graph with $N$ vertices? Explain.

## RUNNING

**54.** Show that if a tree has a vertex of degree $K$, then there are at least $K$ vertices in the tree of degree 1.

**55.** A *bipartite graph* is a graph with the property that the vertices of the graph can be divided into two sets $A$ and $B$ so that every edge of the graph joins a vertex from $A$ to a vertex from $B$ (Fig. 7-43). Explain why trees are always bipartite graphs.

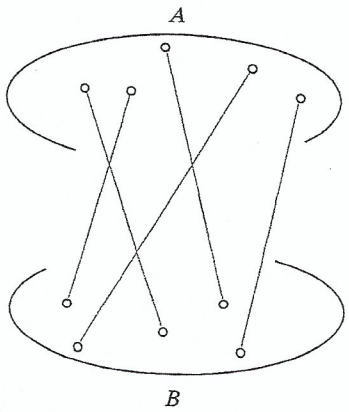



FIGURE 7-43

**56.** Suppose that there is an edge in a network that must be included in any spanning tree. Give an algorithm for finding the minimum spanning tree that includes a given edge. (*Hint*: Modify Kruskal's algorithm.)