Chapter 7: The Mathematics of Networks

7.1 Networks and Trees

Network

Our definition of a network is going to be really simple—essentially, a network is a graph that is connected. In this context the term is most commonly used when the graph models a real-life “network.”

Network

Typically, the vertices of a network (sometimes called nodes or terminals) are “objects” – transmitting stations, computer servers, places, cell phones, people, and so on. The edges of a network (which in this context are often called links) indicate connections among the objects – wires, cables, roads, Internet connections, social connections, and so on.

Network

The World Wide Web is a classic example of an evolutionary network – it follows no predetermined master plan and essentially evolves on its own without structure or centralized direction (yes, there are some rules of behavior but, as we all know, not that many).

Network

At the opposite end of the spectrum from evolutionary networks are networks that are centrally planned and carefully designed to meet certain goals and objectives. Often these types of networks are very expensive to build, and one of the primary considerations when designing such networks is minimizing their cost. This certainly applies to networks of roads, fiber-optic cable lines, rail lines, power lines, and so on.

Optimal Network

The general theme of this chapter is the problem of finding optimal networks connecting a set of points. Optimal means shortest, cheapest, or fastest, depending on whether the cost variable is distance, money, or time. Thus, the design of an optimal network involves two basic goals: (1) to make sure that all the vertices (stations, places, people, etc.) end up connected to the network and (2) to minimize the total cost of the network.
Trees

For obvious reasons, problems of this type are known as minimum network problems. The backbone of a minimum network is a special type of graph called a tree.

This chapter starts with a discussion of the properties of trees.

Example 7.1 The Amazonian Cable Network

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. (How would you maintain and repair the lines otherwise?)

Example 7.1 The Amazonian Cable Network

...and the weight of each edge represents the cost (in millions of dollars) of creating a fiber-optic cable connection along that particular road. The problem facing the engineers and planners at the Amazonia Telephone Company is to build a cable network that (1) utilizes the existing network of roads, (2) connects all the towns, and (3) has the least cost. The challenge, of course, is in meeting the last requirement.

Language of Graphs

Reformulate the three requirements of Example 7.1 in the language of graphs:

1. The network must be a subgraph of the original graph (in other words, its edges must come from the original graph).

2. The network must span the original graph (in other words, it must include all the vertices of the original graph).

3. The network must be minimal (in other words, the total weight of the network should be as small as possible).
The last requirement has an important corollary—a minimal network cannot have any circuits. Why not? Imagine that the solid edges in the figure represent already existing links in a minimal network. Why would you then build the link between X and Y and close the circuit? The edge XY would be a redundant link of the network.

■ A network is just another name for a connected graph. (This terminology is most commonly used when the graph models a real-life situation.) When the network has weights associated to the edges, we call it a weighted network.

■ A network with no circuits is called a tree.

A spanning tree of a network is a subgraph that connects all the vertices of the network and has no circuits.

Among all spanning trees of a weighted network, one with least total weight is called a minimum spanning tree (MST) of the network.

The six graphs on the following slides all have the same set of vertices (A through L). Let’s imagine, for the purposes of illustration, that these vertices represent computer labs at a university, and that the edges are Ethernet connections between pairs of labs.

In this figure, there is no network—the graph is disconnected.

The graph is now connected and we have a network. However, in this network there are several circuits (for example, K, H, I, J, K) that create redundant connections. In other words, this network is not a tree.
Example 7.2 Networks, Trees, and Spanning Trees

In this figure, there is a partial tree connecting some of, but not all, the labs. G and I are left out, so once again, we have no network.

Example 7.2 Networks, Trees, and Spanning Trees

This figure shows a tree that spans (i.e., reaches) all the vertices. We now have a network connecting all the labs and without any redundant connections. Here, the tree is the network.

Example 7.2 Networks, Trees, and Spanning Trees

This figure shows (in red) the same tree as in the previous slide but now highlighted inside of a larger network. In this case we describe the red tree as a spanning tree of the larger network, which shows a different spanning tree for the same network.

Example 7.2 Networks, Trees, and Spanning Trees

This figure shows a different spanning tree for the same network as in the previous slide.

Properties of Trees

As graphs go, trees occupy an important niche between disconnected graphs and “overconnected” graphs. A tree is special by virtue of the fact that it is barely connected. This means several things:

- For any two vertices X and Y of a tree, there is one and only one path joining X to Y. (If there were two different paths joining X and Y, then these two paths would form a circuit, as shown.)
Every edge of a tree is a bridge. (Suppose that some edge \( AB \) is not a bridge. Then without \( AB \) the graph is still connected, so there must be an alternative path from \( A \) to \( B \). This would imply that the edge \( AB \) is part of a circuit as illustrated.)

Among all networks with \( N \) vertices, a tree is the one with the fewest number of edges.

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.)

For \( M = 7 \), the graph becomes connected. Each of these networks is a tree, and thus each of the seven edges is a bridge. Stop here and you will come out $70 richer.

Interestingly, this is as good as it will get. When \( M = 8 \), 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. See the next slide for some examples.
Example 7.3 Connect the Dots (and Then Stop)

As the value of $M$ increases, the number of circuits goes up (very quickly) and the number of bridges goes down.

PROPERTY 1

- In a tree, there is one and only one path joining any two vertices.
- If there is one and only one path joining any two vertices of a graph, then the graph must be a tree.

PROPERTY 2

- In a tree, every edge is a bridge.
- If every edge of a graph is a bridge, then the graph must be a tree.

PROPERTY 3

- A tree with $N$ vertices has $N - 1$ edges.
- If a network has $N$ vertices and $N - 1$ edges, then it must be a tree.

Disconnected Graph

Notice that a disconnected graph (i.e., not a network) can have $N$ vertices and $N - 1$ edges, as shown.
Chapter 7: The Mathematics of Networks

7.2 Spanning Trees, MSTs, and MaxSTs

Example 7.4 Counting Spanning Trees

The network in the figure has $N = 8$ vertices and $M = 8$ edges. The redundancy of the network is $R = M - (N - 1) = 1$, so to find a spanning tree we will have to “discard” one edge.

Example 7.4 Counting Spanning Trees

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.

Example 7.4 Counting Spanning Trees

Thus, the network has three different spanning trees.

Example 7.4 Counting Spanning Trees

The network in the figure has $M = 9$ edges and $N = 8$ vertices. 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.
Example 7.4 Counting Spanning Trees

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\}.

Example 7.4 Counting Spanning Trees

For example, if we exclude BC and CD, we get the spanning tree shown.

Example 7.4 Counting Spanning Trees

We could also exclude BC and DE and get the spanning tree shown.

Example 7.4 Counting Spanning Trees

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, there are 12 spanning trees. We have already shown two, here’s one more.

Example 7.5 Counting Spanning Trees

This network has \(M = 9\) edges and \(N = 8\) vertices. 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.

Example 7.6 Minimum Spanning Trees

This example builds on the ideas introduced in Example 7.2. The network shown in is the same network as Example 7.2 now with weights added to the edges.
Vertices represent computer labs, and edges are potential Ethernet connections. The weights represent the cost in K’s (1K = $1000) of installing the Ethernet connections.

Example 7.6 Minimum Spanning Trees

Using the terminology we introduced in this section, the weighted network has a redundancy of R = 3 (the network has M = 14 vertices and N = 12 edges).

Example 7.6 Minimum Spanning Trees

The network has many possible spanning trees, and our job is to find one with least weight, that is, a minimum spanning tree (MST) for the network. We know that we can find a spanning tree by excluding certain edges of the network (those that close circuits) from the spanning tree and that there are many different ways in which this can be done. Given that the edges now have weights, a reasonable strategy for sorting through the options would be to always try to exclude from the network the most expensive edges.

Likewise, on the right side of the network we have a configuration of two circuits K, H, I, K and K, J, I, K that share the common edge KI.

We exclude the two most expensive edges from these two circuits (KJ and KI).
At this point, all the circuits of the original network are “busted,” and we end up with the red spanning tree shown.

Is the red spanning tree obtained in Example 7.6 the MST of the original network? We would like to think it is, but how can we be sure, especially after the lessons learned in Chapter 6? And even if it is, what assurances do we have that our simple strategy will work in more complicated graphs? These are the questions we will answer next.

Kruskal’s algorithm is almost identical to the cheapest-link algorithm: We build the minimum spanning tree one edge at a time, choosing at each step the cheapest available edge. The only restriction to our choice of edges is that we should never choose an edge that creates a circuit. (Having three or more edges coming out of a vertex, however, is now OK.) What is truly remarkable about Kruskal’s algorithm is that—unlike the cheapest-link algorithm—it always gives an optimal solution.

What is the optimal fiber-optic cable network connecting the seven towns shown?

The weighted graph shows the costs (in millions of dollars) of laying the cable lines along each of the potential links of the network.
Example 7.7 The Amazonian Cable Network: Part 2

The answer, as we now know, is to find the MST of the graph. We will use Kruskal’s algorithm to do it. Here are the details:

Step 1. Among all the possible links, we choose the cheapest one, in this particular case GF (at a cost of $42 million). This link is going to be part of the MST, and we mark it in red (or any other color) as shown.

Step 2. The next cheapest link available is BD at $45 million. We choose it for the MST and mark it in red.

Step 3. The next cheapest link available is AD at $49 million. Again, we choose it for the MST and mark it in red.

Step 4. For the next cheapest link 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! The link DG, on the other hand, is just fine, so we mark it in red and make it part of the MST.

Step 5. The next cheapest link 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 link available is BC at $55 million, but this link 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. The figure shows the MST in red. The total cost of the network is $299 million.
Step 1. Pick the cheapest link available. (In case of a tie, pick one at random.) Mark it (say in red).

Step 2. Pick the next cheapest link available and mark it.

Steps 3, 4, ..., N – 1. Continue picking and marking the cheapest unmarked link available that does not create a circuit.

As algorithms go, Kruskal’s algorithm is as good as it gets. First, it is easy to implement. Second, it is an efficient algorithm. As we increase the number of vertices and edges of the network, the amount of work grows more or less proportionally (roughly speaking, if finding the MST of a 30-city network takes you, say, 30 minutes, finding the MST of a 60-city network might take you 60 minutes).

Last, but not least, Kruskal’s algorithm is an optimal algorithm—it will always find a minimum spanning tree. Thus, we have reached a surprisingly satisfying end to the MST story: No matter how complicated the network, we can find its minimum spanning tree by means of an algorithm that is easy to understand and implement, is efficient, and is also optimal.