7.5 Trees

We now discuss a special type of graph called a *tree*. We start with an example that shows why these graphs are useful.

Sample Problem 7.14. *Suppose an oil field contains five oil wells w_1, w_2, w_3, w_4, w_5, and a depot d. It is required to build pipelines so that oil can be pumped from the wells to the depot. Oil can be relayed from one well to another, at very small cost. The only major expense is building the pipelines. Figure* 7.16(a) *shows which pipelines are feasible to build, represented as a graph in the obvious way, with the cost* (*in hundreds of thousands of dollars*) *shown* (*a missing edge might mean that the cost would be very high*)*. Which pipelines should be built?*

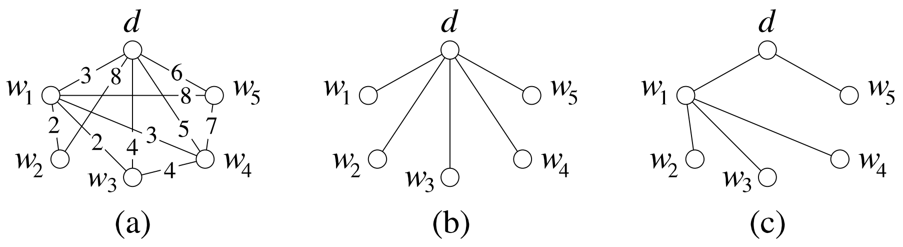


Fig. 7.16. An oil pipeline problem

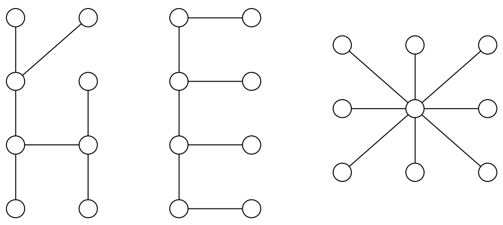


Fig. 7.17. Some trees

Solution. Your first impulse might be to connect d to each well directly, as shown in Figure 7.16(b). This would cost \$2,600,000. However, the cheapest solution is shown in Figure 7.16(c), and costs \$1,600,000.

Once the pipelines in case (c) have been built, there is no reason to build a direct connection from d to w_3. Any oil being sent from w_3 to the depot can be relayed through w_1. Similarly, there would be no point in building a pipeline joining w_3 to w_4. The conditions of the problem imply that the graph does not need to contain any cycle. However, it must be connected.

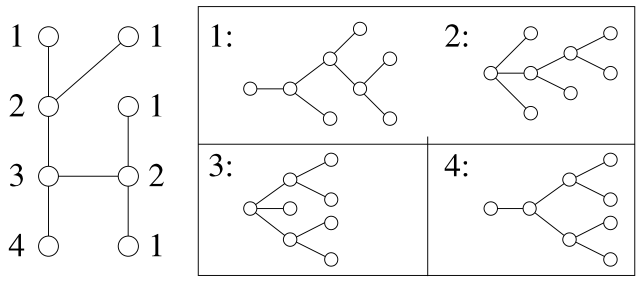
A connected graph that contains no cycle is called a *tree*. So the solution of the problem is to find a subgraph of the underlying graph that is a tree, and among those to find the one that is cheapest to build. We shall return to this topic shortly. But first we look at trees as graphs.

We have already seen trees. The tree diagrams used in Chapter 6 are just trees laid out in a certain way. To form a tree diagram from a tree, select any vertex as a starting point (called the *root*). Draw edges to all of its neighbors. These are the first generation. Then treat each neighbor as if it were the starting point; draw edges to all its neighbors except the start. Continue in this way; to each vertex, attach all of its neighbors except for those that have already appeared in the diagram.

Figure 7.17 contains three examples of trees. It is also clear that every path is a tree, and the star $K_{1,n}$ is a tree for every n.

Sample Problem 7.15. *Draw the first tree in Figure 7.17 as a tree diagram. How many different-looking diagrams can be made?*

Solution. There are four different-looking diagrams. In the following illustration, the vertices of the original tree are labeled 1, 2, 3, 4. The diagrams obtained using each type of vertex as root are shown.



Practice Exercise. Repeat this problem for the other two trees in Figure 7.17.

A tree is a minimal connected graph in the following sense: if any vertex of degree at least 2, or any edge, is deleted, then the resulting graph is not connected. In fact, it is easy to prove that a connected graph is a tree if and only if every edge is a bridge.

Trees can also be characterized among connected graphs by their number of edges.

Theorem 48. *A finite connected graph G with v vertices is a tree if and only if it has exactly $v - 1$ edges.*

Proof. (i) Suppose that G is a tree with v vertices. We proceed by induction on v. The theorem is true for $v = 1$ since the only graph with one vertex is K_1, which is a tree. Suppose it is true for $w < v$, and suppose G is a tree with v vertices. Select an edge (G must have an edge, or it will be the unconnected graph $\overline{K}_v$) and delete it. The result is a union of two disjoint components, each of which is a tree with fewer than v vertices; say the first component has v_1 vertices and the second has v_2, where $v_1 + v_2 = v$. By the induction hypothesis, these graphs have $v_1 - 1$ and $v_2 - 1$ edges, respectively. Adding one edge for the one that was deleted, we find that the number of edges in G is

$$(v_1 - 1) + (v_2 - 1) + 1 = v - 1.$$

(ii) Conversely, suppose G is not a tree. Select an edge that is *not* a bridge and delete it. If the resulting graph is not a tree, repeat the process. Eventually there will be only bridges left and the graph is a tree. From what we have just said it must have $v - 1$ edges, and the original graph had more than $v - 1$ edges. □

Corollary 2. Every tree other than K_1 has at least two vertices of degree 1.

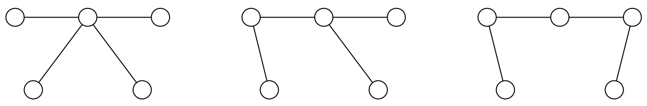
Proof. Suppose the tree has v vertices. It then has $v-1$ edges. So, by Theorem 43, the sum of all degrees of the vertices is $2(v-1)$. There can be no vertex of degree 0 since the tree is connected and is not K_1; if $v-1$ of the vertices have degree at least 2, then the sum of the degrees is at least $1+2(v-1)$, which is impossible. □

Sample Problem 7.16. *Find all trees with five vertices.*

Solution. If a tree has five vertices, then the largest possible degree is 4. Moreover there are four edges (by Theorem 48), so from Theorem 43 the sum of the five degrees is 8. As there are no vertices of degree 0 and at least two vertices of degree 2, the list of degrees must be one of

$$4, 1, 1, 1, 1, \quad 3, 2, 1, 1, 1, \quad 2, 2, 2, 1, 1.$$

In the first case, the only solution is the star $K_{1,4}$. If there is one vertex of degree3, no two of its neighbors can be adjacent (this would form a cycle), so the fourth edge must join one of those three neighbors to the fifth vertex. The only case with the third degree list is the path P_5. These three cases are as follows:



Practice Exercise. Find all trees with four vertices.

Spanning Trees; The Minimal Spanning Tree

A *spanning* subgraph of a graph G is one that contains every vertex of G. A *spanning tree* in a graph is a spanning subgraph that is a tree when considered as a graph in its own right. For example, it was essential that the tree used to plan the oil pipelines should be a spanning tree.

It is clear that any connected graph G has a spanning tree. If G is a tree, then the whole of G is itself the spanning tree. Otherwise G must contain a cycle; delete one edge from the cycle. The resulting graph is still a connected subgraph of G; and, as no vertex has been deleted, it is a spanning subgraph. Find a cycle in this new graph and delete it; repeat the process. Eventually the remaining graph will contain no cycle, so it is a tree. So when the process stops, we have found a spanning tree. A given graph might have many different spanning trees.

In many applications—such as the oil pipeline problem—each edge of a graph has a weight associated with it, such as distance or cost. It is sometimes desirable to find a spanning tree such that the weight of the tree—the total of the weights of its edges—is minimum. Such a tree is called a *minimal spanning tree*.

A finite graph can contain only finitely many spanning trees, so it is possible in theory to list all spanning trees and their weights, and to find a minimal spanning tree by choosing one with minimum weight. This process could take a very long time, however, since the number of spanning trees in a graph can be very large. So efficient algorithms that find a minimal spanning tree are useful. We present here an example due to Prim.

We assume that G is a graph with vertex set V and edge set E, and suppose there is associated with G a map $w : E \to R$ called the *weight* of the edge; when xy is an edge of G we write $w(x, y)$ for the image of xy under w. We can quite easily modify the algorithm to allow for multiple edges, but the notation is slightly simpler in the graph case. The algorithm consists of finding a sequence of vertices $x_0, x_1, x_2, \dots,$ of G and a sequence of sets $S_0, S_1, S_2, \dots,$ where

$$S_i = \{x_0, x_1, \dots, x_{i-1}\}.$$

We choose x_0 at random from V. When $n > 0$, we find x_n inductively using S_n as follows.

1. Given $i, 0 \le i \le n-1$, choose y_i to be a member of $V \setminus S_n$ such that $w(x_i, y_i)$ is minimum, if possible.

2. Provided that at least one y_i has been found in step (1), define x_n to be a y_i such that $w(x_i, y_i)$ is minimal.

3. Put $S_{n+1} = S_n \cup \{x_n\}$.

This process stops only when there is no new vertex y_i. If there is no new vertex y_i, it must be true that no member of $V \setminus S_n$ is adjacent to a vertex of S_n. It is impossible to partition the vertices of a connected graph into two nonempty sets such that no edge joins one set to the other, and S_n is never empty, so the process stops only when $S_n = V$.

When we reach this stage, so that $S_n = V$, we construct a graph T as follows:

(i) T has vertex set V;

(ii) if x_k arose as y_i, then x_i is adjacent to x_k in T;

(iii) no edges of T exist other than those that may be found using (ii).

It is not hard to verify that T is a tree and that it is minimal.

Observe that X_n may not be defined uniquely at step (2) of the algorithm, and indeed y_i may not be uniquely defined. This is to be expected: after all, there may be more than one minimal spanning tree.

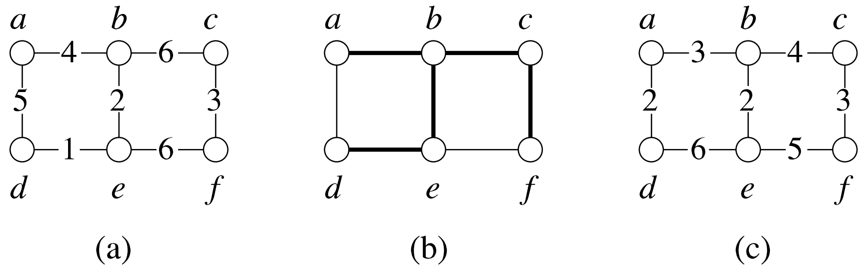


Fig. 7.18. An example of Prim's algorithm

Sample Problem 7.17. *Find a minimal spanning tree in the graph G shown in Figure* 7.18(a).

Solution. Select $x_0 = a$. Then $S_1 = \{a\}$. Now $y_0 = b$, and this is the only choice for x_1. So $S_2 = \{a, b\}$. The tree will contain edge ab.

Working from S_2, we get $y_0 = d$ and $y_1 = e$. Since be $(= x_1y_1)$ has smaller weight than ad $(= x_0y_0)$, we select $x_2 = e$. Then $S_3 = \{a, b, e\}$ and edge be goes into the tree. Similarly, from S_3, we get $x_3 = d$ and $S_4 = \{a, b, d, e\}$, and the new edge is de.

Now there is a choice. Working from S_4, $y_1 = c$ and $y_2 = f$. In both cases the weight of the edge to be considered is 6. So either may be used. Let us choose c and use edge bc.

The final vertex is f, and the edge is cf. So the tree has edges ab, bc, be, cf, de, and weight 16. It is shown in Figure 7.18(b).

Practice Exercise. Find a minimal spanning tree in the graph G shown in Figure 7.18(c).

The algorithm might be described as follows. First, choose a vertex x_0. Trivially the minimum weight tree with vertex-set $\{x_0\}$—the *only* tree with vertex-set $\{x_0\}$—is the K_1 with vertex x_0. Call this the *champion*. Then find the smallest weight tree with two vertices, one of which is x_0; in other words, find the minimum weight tree that can be formed by adding just one edge to the current champion. This tree is the new champion. Continue in this way: each time a champion is found, look for the cheapest tree that can be formed by adding one edge to it. One can consider each new tree to be an approximation to the final minimal spanning tree, with successive approximations having more and more edges.

Prim's algorithm was a refinement of an earlier algorithm due to Kruskal. In that algorithm, one starts by listing all edges in order of increasing weight. The first approximation is the K_2 consisting of the edge of least weight. The second approximation is formed by appending the next edge in the ordering. At each stage the next approximation is formed by adding on the smallest edge that has not been used, provided only that it does not form a cycle with the edges already chosen. In this case the successive approximations are not necessarily connected until the last one. The advantage of Prim's algorithm is that, in large graphs, the initial sorting stage of Kruskal's algorithm can be very time consuming.