7.2 Minimum-Cost Spanning Trees

Suppose *G* = (*V*, *E*) is a connected graph in which each edge (*u*, *v*) in *E* has a cost *c*(*u*, *v*) attached to it. A *spanning tree* for *G* is a free tree that connects all the vertices in *V*. The *cost* of a spanning tree is the sum of the costs of the edges in the tree. In this section we shall show how to find a minimum-cost spanning tree for *G*.

Example 7.4. Figure 7.4 shows a weighted graph and its minimum-cost spanning tree.

A typical application for minimum-cost spanning trees occurs in the design of communications networks. The vertices of a graph represent cities and the edges possible communications links between the cities. The cost associated with an edge represents the cost of selecting that link for the network. A minimum-cost spanning tree represents a communications network that connects all the cities at minimal cost.

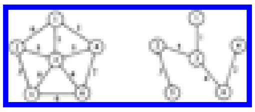

Fig. 7.4. A graph and spanning tree.

The MST Property

There are several different ways to construct a minimum-cost spanning tree. Many of these methods use the following property of minimum-cost spanning trees, which we call the *MST property*. Let *G* = (*V*, *E*) be a connected graph with a cost function defined on the edges. Let *U* be some proper subset of the set of vertices *V*. If (*u*, *v*) is an edge of lowest cost such that *u* ∈ *U* and *v* ∈ *V-U*, then there is a minimum-cost spanning tree that includes (*u*, *v*) as an edge.

The proof that every minimum-cost spanning tree satisfies the MST property is not hard. Suppose to the contrary that there is no minimum-cost spanning tree for *G* that includes (*u*, *v*). Let *T* be any minimum-cost spanning tree for *G*. Adding (*u*, *v*) to *T* must introduce a cycle, since *T* is a free tree and therefore satisfies property (2) for free trees. This cycle involves edge (*u*, *v*). Thus, there must be another edge (*u'*, *v'*) in *T* such that *u'* ∈ *U* and *v'* ∈ *V-U*, as illustrated in Fig. 7.5. If not, there would be no way for the cycle to get from *u* to *v* without following the edge (*u*, *v*) a second time.

Deleting the edge (*u'*, *v'*) breaks the cycle and yields a spanning tree *T'* whose cost is certainly no higher than the cost of *T* since by assumption $c(u, v) \le c(u', v')$. Thus, *T'* contradicts our assumption that there is no minimum-cost spanning tree that includes (*u*, *v*).

Prim's Algorithm

There are two popular techniques that exploit the MST property to construct a minimum-cost spanning tree from a weighted graph *G* = (*V*, *E*). One such method is known as Prim's algorithm. Suppose *V* = {1, 2, . . . , *n*}. Prim's algorithm begins with a set *U* initialized to {1}. It then "grows" a spanning tree, one edge at a time. At each step, it finds a shortest edge (*u*, *v*) that connects *U* and *V-U* and then adds *v*, the vertex in *V-U*, to *U*. It repeats this

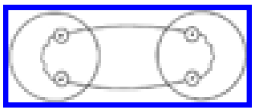

Fig. 7.5. Resulting cycle.

step until *U* = *V*. The algorithm is summarized in Fig. 7.6 and the sequence of edges added to *T* for the graph of Fig. 7.4(a) is shown in Fig. 7.7.

```
procedure Prim ( G: graph; var T: set of edges );
  { Prim constructs a minimum-cost spanning tree T for G }
  var
      U: set of vertices;
      u, v: vertex;
  begin
     T:= Ø;
     U := {1};
     while U ≠ V do begin
          let (u, v) be a lowest cost edge such that
               u is in U and v is in V-U;
        T := T ∪ {(u, v)};
        U := U ∪ {v}
     end
 end; { Prim }
```

Fig. 7.6. Sketch of Prim's algorithm.