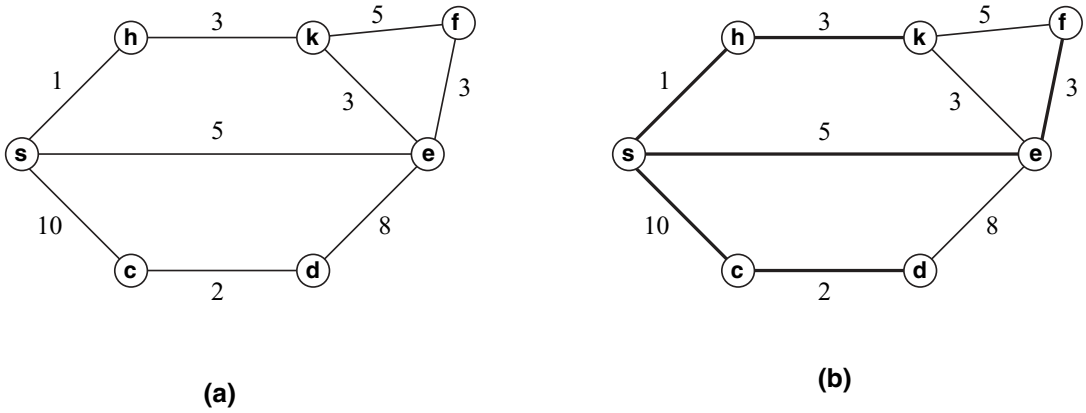

FIGURE 4.1: Determining the shortest paths from s to the other entities.

the problem of maintaining the information of the tables up to date, should changes occur in the system. Finally, we will discuss how to represent routing information in a compact way, suitable for systems where space is a problem. In the following, and unless otherwise specified, we will assume the set of restrictions **IR**: *Bidirectional Links* (BL), *Connectivity* (CN), *Total Reliability* (TR), and *Initial Distinct Values* (ID).

4.2 SHORTEST PATH ROUTING

The *routing table* of an entity contains information on how to reach any possible destination. In this section we examine how this information can be acquired, and the table constructed. As we will see, this problem is related to the construction of particular spanning-trees of the network. In the following, and unless otherwise specified, we will focus on shortest-path routing.

Different types of routing tables can be defined, depending on the amount of information contained in them. We will consider for now the *full* routing table: For each destination, there is stored a shortest path to reach it; if there are more than one shortest path, only the lexicographically smallest[1] will be stored. For example, in the network of Figure 4.1, the routing table $RT(s)$ for s is shown in Table 4.1.

We will see different approaches to construct routing tables, some depending on the amount of local storage an entity has available.

4.2.1 Gossiping the Network Maps

A first obvious solution would be to construct at every entity the entire *map* of the network with all the costs; then, each entity can locally and directly compute its shortest-path routing table. This solution obviously requires that the local memory available to an entity is large enough to store the entire map of the network.

[1] The lexicographic order will be over the strings of the names of the nodes in the paths.

TABLE 4.1: Full Routing Table for Node *s*

Routing Destination	Shortest Path	Cost
h	(s, h)	1
k	$(s, h)(h, k)$	4
c	(s, c)	10
d	$(s, c)(c, d)$	12
e	(s, e)	5
f	$(s, e)(e, f)$	8

The map of the network can be viewed as an $n \times n$ array MAP(G), one row and one column per entity, where for any two entities x and y, the entry MAP$[x, y]$ contains information on whether link (x, y) exists, and if so on its cost. In a sense, each entity x knows initially only its own row MAP$[x, \star]$. To know the entire map, every entity needs to know the initial information of all the other entities.

This is a particular instance of a general problem called *input collection* or *gossip*: every entity has a (possibly different) piece of information; the goal is to reach a final configuration where every entity has all the pieces of information. The solution of the gossiping problem using normal messages is simple:

every entity broadcasts its initial information.

Since it relies solely on broadcast, this operation is more efficiently performed in a tree. Thus, the protocol will be as follows:

Map_Gossip:

1. An arbitrary spanning tree of the network is created, if not already available; this tree will be used for all communication.
2. Each entity acquires full information about its neighborhood (e.g., names of the neighbors, cost of the incident links, etc.), if not already available.
3. Each entity broadcasts its neighborhood information along the tree.

At the end of the execution, each entity has a complete map of the network with all the link costs; it can then locally construct its shortest-path routing table.

The construction of the initial spanning-tree can be done using $O(m + n \log n)$ messages, for example using protocol *MegaMerger*. The acquisition of neighborhood information requires a single exchange of messages between neighbors, requiring in total just $2m$ messages. Each entity x then broadcasts on the tree $\deg(x)$ items of information. Hence the total number of messages will be at most

$$\sum_x \deg(x)(n-1) = 2m(n-1).$$

Thus, we have

$$\mathbf{M}[Map\_Gossip] = 2\,m\,n + \textit{l.o.t.} \tag{4.1}$$