ROUTING ALGORITHMS: SHORTEST PATH AND WIDEST PATH

Lecture 3

INTRODUCTION

Shortest path algorithms are applicable to IP networks and widest path algorithms are useful for telephone network dynamic call routing.

They appear in network routing in many ways and have played critical roles in the development of routing protocols.

A communication network is made up of nodes and links.

In an IP network, a node is called a *router* while in the telephone network a node is either an *end (central) office* or a *toll switch.*

In an optical network, a node is an *optical or electro-optical switch.*

INTRODUCTION

A link connects two nodes; a link connecting two routers in an IP network is sometimes called an *IP trunk* or simply an IP link, while the end of a link outgoing from a router is called an *interface.*

A link in a telephone network is called a *trunk group*, or an *intermachine trunk (IMT)*, and sometimes simply a *trunk*.

A communication network carries traffic where traffic flows from a *start* node to an *end* node; typically, we refer to the start node as the *source* node (where traffic originates) and the end node as the *destination* node.

INTRODUCTION

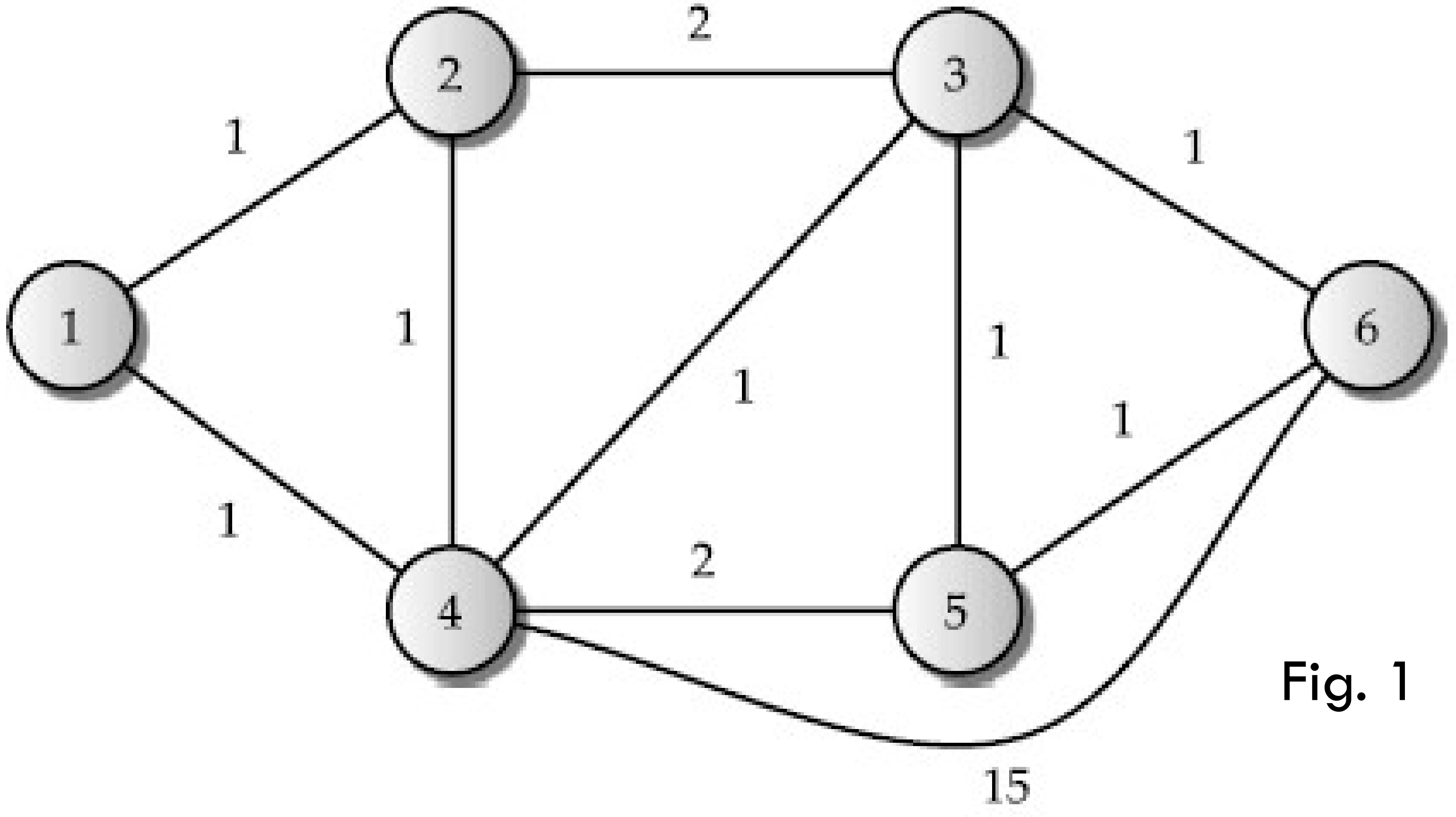


Fig. 1

INTRODUCTION

Consider now the network shown in Figure 1.

Suppose that we have traffic that enters node 1 destined for node 6; in this case, node 1 is the source node and node 6 is the destination node.

We may also have traffic from node 2 to node 5; for this case, the source node will be node 2 and the destination node will be node 5; and so on.

An important requirement of a communication network is to flow or *route* traffic from a source node to a destination node.

To do that we need to determine a route, which is a path from the source node to the destination node.

A route can certainly be set up manually; such a route is known as a *static route*.

INTRODUCTION

However, it is desirable to use a routing *algorithm* to determine a route.

The **goal** of a routing algorithm is in general dictated by the requirement of the communication network and the service it provides as well as any additional or specific goals a service provider wants to impose on itself.

While goals can be different for different types of communication networks, they can usually be classified into two broad categories: user oriented and network-oriented.

User-oriented means that a network needs to provide good service to each user so that traffic can move from the source to the destination quickly for this user.

This should not be for a specific user at the expense of other users between other source–destination nodes in the network.

INTRODUCTION

Thus, a network's goal ("network-oriented") generally is to address how to provide an efficient and fair routing so that *most* users receive good and acceptable service, instead of providing the "best" service to a specific user.

Such a view is partly required because there are a finite amount of resources in a network, e.g., network capacity.

Consider two very important algorithms that have profound impact on data networks, in particular on Internet routing.

These two algorithms, known as the *Bellman–Ford algorithm* and *Dijkstra's algorithm*, can be classified as falling under user-oriented in terms of the above broad categories.

INTRODUCTION

They are both called shortest path routing algorithms, i.e., an algorithm where the goal is to find the shortest path from a source node to a destination node.

A simple way to understand a shortest path is from road networks where shortest can be defined in terms of distance, for example, as in what is the shortest distance between two cities, which consists of the link distance between appropriate intermediate places between the end cities.

However, it is possible that notions other than the usual distance-based measure may be applicable as well, for instance, time taken to travel between two cities.

In other words, an equally interesting question concerns the shortest route between two cities in terms of *time*.

INTRODUCTION

Instead of worrying about the unit of measurement, it is better to have an algorithm that works *independent* of the measuring unit and considers a generic measure for distance for each link in a network.

In communication networks, a generic term to refer to a distance measure without assigning any measure units is called *cost*, *link cost*, *distance cost*, or *link metric*.

Consider again Figure 1. We have assigned a value with each link, e.g., link 4-6 has the value 15; we will say that the link cost, or distance cost, or link metric of link 4-6 is 15.

No measuring units are used; for example, in road networks, it could be in miles, kilometers, or minutes. By simple inspection, it is not hard to see that the shortest path between nodes 1 and 6 is the path 1-4-3-6 with a total minimum cost of 3.

INTRODUCTION

It may be noted that the shortest path in this case did not include the link 4-6, although from the viewpoint of the number of nodes visited, it would look like the path 1-4-6 is the shortest path between nodes 1 and 6.

In fact, this would be the case if the link cost was measured in terms of nodes visited, or *hops*. In other words, if the number of hops is important for measuring distance for a certain network, we can then think about the network in Figure 2.1 by considering the link cost for each link to be 1 instead of the number shown on each link in the figure.

Regardless, having an algorithm that works without needing to worry about how cost is assigned to each link is helpful; this is where the Bellman–Ford and Dijkstra's algorithms both fit in.

INTRODUCTION

At this point, it is important to point out that in computing the shortest path, the *additive* property is generally used for constructing the overall distance of a path by adding a cost of a link to the cost of the next link along a path until all links for the path are considered.

A network can be expressed as a graph by mapping each node to a unique vertex in the graph where links between network nodes are represented by edges connecting the corresponding vertices.

Each edge can carry one or more weights; such weights may depict cost, delay, bandwidth, and so on.

Figure 1 depicts a network consisting of a graph of six nodes and ten links where each link is assigned a link cost/weight.

BELLMAN–FORD ALGORITHM AND THE DISTANCE VECTOR APPROACH

The Bellman–Ford algorithm uses a simple idea to compute the shortest path between two nodes in a centralized fashion.

In a distributed environment, a distance vector approach is taken to compute shortest paths.

We will discuss the centralized approach.

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

We will use two generic nodes, labeled as node *i* and node *j*, in a network of *N* nodes. They may be directly connected as a link such as link 4-6 with end nodes 4 and 6 (see Figure 1).

As can be seen from Figure 1, many nodes are not directly connected, for example, nodes 1 and 6; in this case, to find the distance between these two nodes, we need to resort to using other nodes and links.

This brings us to an important point; we may have the notion of cost between two nodes, irrespective of whether they are directly connected or not. Thus, we introduce two important notations:

d_{ij} = Link cost between nodes *i* and *j*

$\overline{D_{ij}}$ = Cost of the computed minimum cost path from node *i* to node *j*.

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

Since we are dealing with different algorithms, we will use overbars, underscores, and hats in our notations to help distinguish the computation for different classes of algorithms.

For example, overbars are used for all distance computation related to the Bellman–Ford algorithm and its variants.

If two nodes are directly connected, then the link cost d_{ij} takes a finite value.

Consider again Figure 1. Here, nodes 4 and 6 are directly connected with link cost 15; thus, we can write $d_{46} = 15$.

On the other hand, nodes 1 and 6 are not directly connected; thus, $d_{16} = \infty$.

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

What then is the difference between d_{ij} and the minimum cost D_{ij}?

From nodes 4 to 6, we see that the minimum cost is actually 2, which takes path 4-3-6; that is, $\overline{D_{46}} = 2$ while $d_{46} = 15$.

For nodes 1 and 6, we find that $\overline{D_{16}} = 3$ while $d_{16} = \infty$.

As can be seen, a minimum cost path can be obtained between two nodes in a network regardless of whether they are directly connected or not, as long as one of the end nodes is not completely isolated from the rest of the network.

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

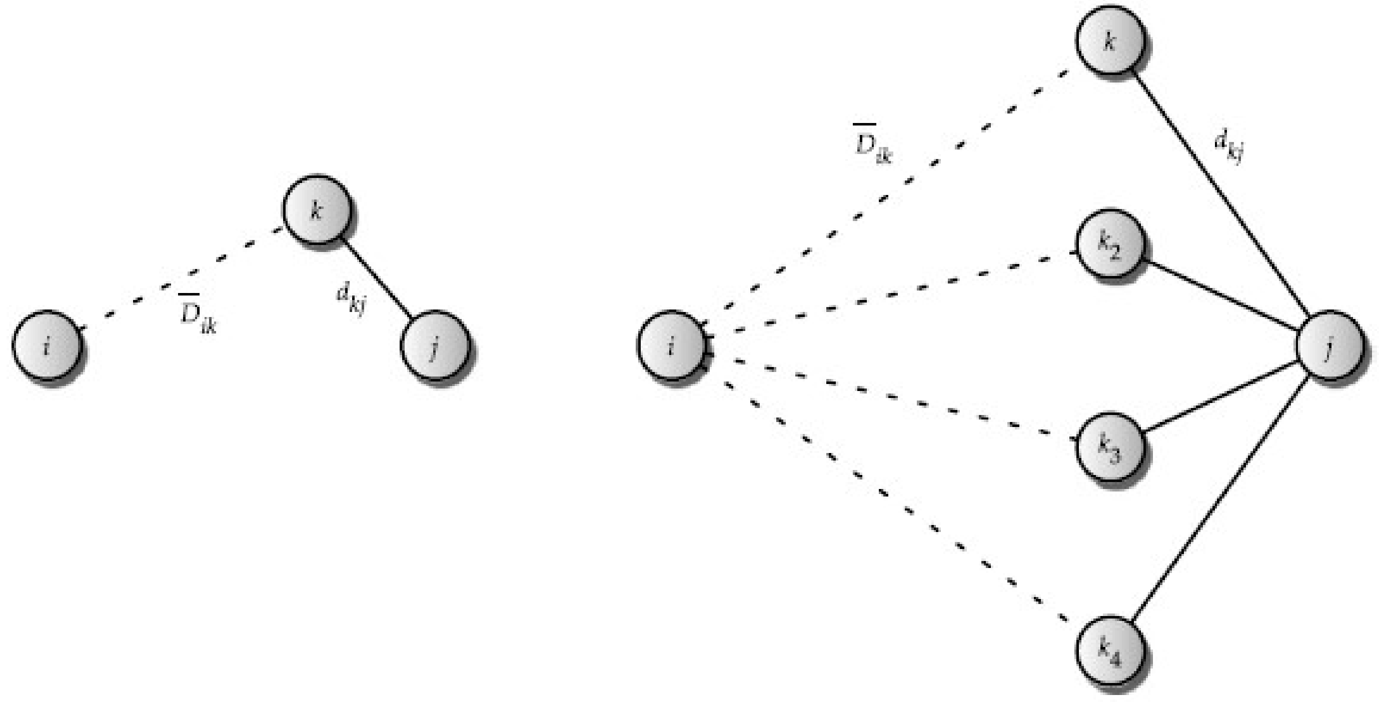


Fig. 2: Centralized Bellman–Ford Algorithm (solid line denotes a direct link; dashed line denotes distance).

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

The question now is how to compute the minimum cost between two nodes in a network.

This is where shortest path algorithms come in.

To discuss such an algorithm, it is clear from the six-node example that we also need to rely on intermediate nodes.

For that, consider a generic node *k* in the network that is directly connected to either of the end nodes; we assume that *k* is directly connected to the destination node *j*, meaning d_{kj} has a finite value.

The following equations, known as Bellman's equations, must be satisfied by the shortest path from node *i* to node *j*:

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

$\overline{D_{ii}} = 0, for\ all\ i$ (1a)

$\overline{D_{ij}} = min_{k \neq j}\{\overline{D_{ik}} + d_{kj}\}, for\ all\ i \neq j$ (1b)

Simply put, Eq. (1b) states that for a pair of nodes *i* and *j*, the minimum cost is dependent on knowing the minimum cost from *i* to *k* and the direct link cost d_{kj} for link *k*-*j*.

A visual is shown in Figure 2.

Note that there can be multiple nodes *k* that can be directly connected to the end node *j* (say they are marked *k*, k_2, and so on; note that $k = i$ is not ruled out either); thus, we need to consider all such *k*s to ensure that we can compute the minimum cost.

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

It is important to note that technically, a node *k* that is not directly connected to *j* is also considered; since for such *k*, we have $d_{kj} = \infty$, the resulting minimum computation is not impacted.

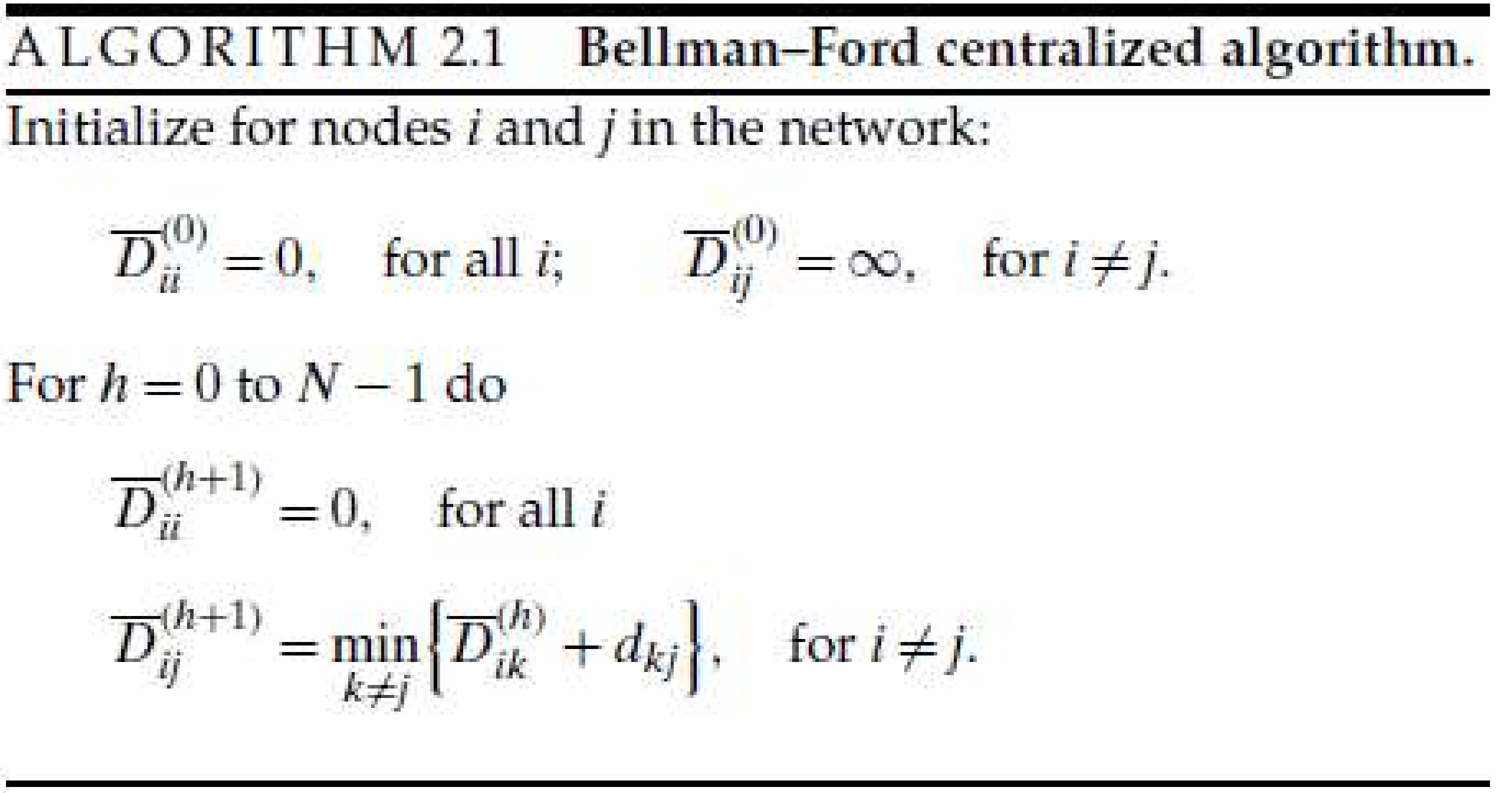
On close examination, note that Eq. (1b) assumes that we know the minimum cost, $\overline{D_{ik}}$, from node *i* to *k* first somehow!

Thus, in an actual algorithmic operation, a slight variation of Eq. (1b) is used where the minimum cost is accounted for by iterating through the number of hops.

Specifically, we define the term for the minimum cost in terms of number of hops *h* as follows:

$\overline{\boldsymbol{D_{ij}^{h}}} =$ **cost of the minimum cost path from node *i* to node *j* when up to *h* number of hops are considered.**

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM



ALGORITHM 2.1 Bellman–Ford centralized algorithm.

```
Initialize for nodes $i$ and $j$ in the network:
    $\overline{D}_{ii}^{(0)} = 0$, for all $i$;    $\overline{D}_{ij}^{(0)} = \infty$, for $i \neq j$.    2a
For $h = 0$ to $N - 1$ do
    $\overline{D}_{ii}^{(h+1)} = 0$, for all $i$    2b
    $\overline{D}_{ij}^{(h+1)} = \min_{k \neq j} \left\{ \overline{D}_{ik}^{(h)} + d_{kj} \right\}$, for $i \neq j$.    2c
```

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

The Bellman–Ford algorithm that iterates in terms of number of hops is given in Algorithm 2.1.

Note the use of *(h)* in the superscript in Eq. (2c); while the expression on the right side is up to *h* hops, with the consideration of one more hop, the expression on the left hand side is now given in *h* +1 hops.

For the six-node network (Figure 1), the Bellman–Ford algorithm is illustrated in Table 2.1.

A nice way to understand the hop-iterated Bellman–Ford approach is to visualize through an example.

Consider finding the shortest path from node 1 to node 6 as the number of hops increases.

CENTRALIZED VIEW: BELLMAN—FORD ALGORITHM

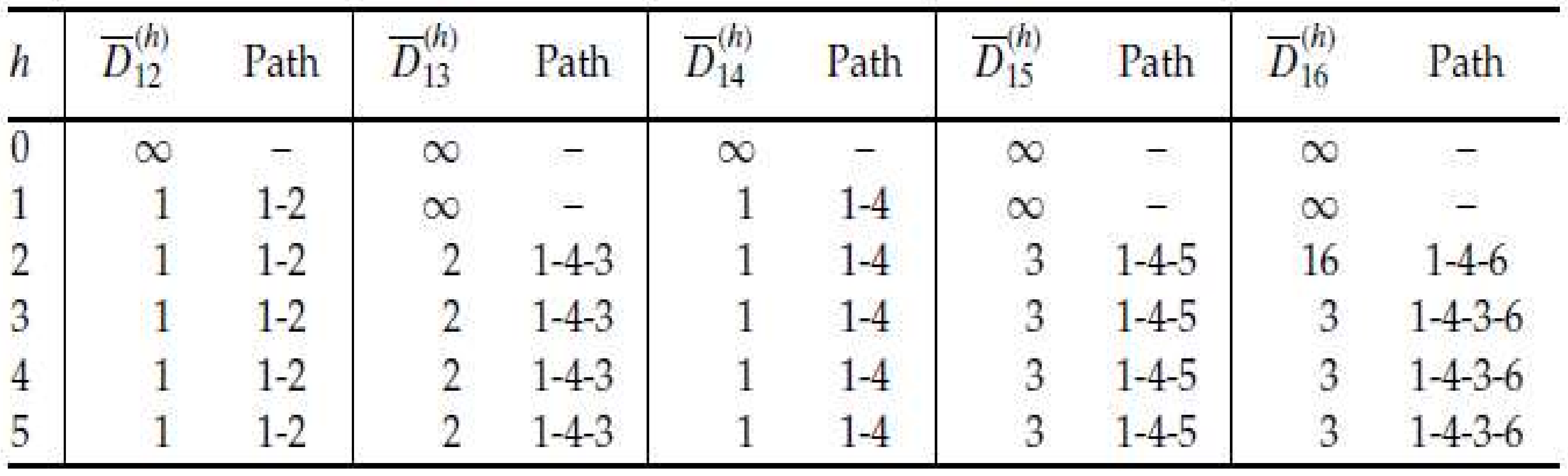


TABLE 2.1 Minimum cost from node 1 to other nodes using Algorithm 2.1.

h	$\overline{D}_{12}^{(h)}$	Path	$\overline{D}_{13}^{(h)}$	Path	$\overline{D}_{14}^{(h)}$	Path	$\overline{D}_{15}^{(h)}$	Path	$\overline{D}_{16}^{(h)}$	Path
0	∞	–	∞	–	∞	–	∞	–	∞	–
1	1	1-2	∞	–	1	1-4	∞	–	∞	–
2	1	1-2	2	1-4-3	1	1-4	3	1-4-5	16	1-4-6
3	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6
4	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6
5	1	1-2	2	1-4-3	1	1-4	3	1-4-5	3	1-4-3-6

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

When $h = 1$, it means considering a direct link path between 1 and 6; since there is none, $D_{16}^{(1)} = \infty$.

With $h = 2$, the path 1-4-6 is the only one possible since this is a two-link path, i.e., it uses two hops, consisting of the links 1-4 and 4-6; in this case, the hop-iterated minimum cost is 16 (= $D_{16}^{(2)}$). At $h = 3$, we can write the Bellman–Ford step as follows (shown only for k for which $d_{k6} < \infty$) since there are three possible paths that need to be considered:

$$
\begin{aligned}
k = 3: &\quad \overline{D}_{13}^{(2)} + d_{36} = 2 + 1 = 3 \\
k = 5: &\quad \overline{D}_{15}^{(2)} + d_{56} = 3 + 1 = 4 \\
k = 4: &\quad \overline{D}_{14}^{(2)} + d_{46} = 1 + 15 = 16.
\end{aligned}
$$

CENTRALIZED VIEW: BELLMAN–FORD ALGORITHM

In this case, we pick the first one since the minimum cost is 3, i.e., $\overline{D_{16}^{(3)} = 3}$ with the shortest path 1-4-3-6.

It is important to note that the Bellman–Ford algorithm computes only the minimum cost; it does not track the actual shortest path.

We have included the shortest path in Table 2.1 for ease of understanding how the algorithm works.

For many networking environments, it is not necessary to know the entire path; just knowing the next node *k* for which the cost is minimum is sufficient—this can be easily tracked with the min operation in Eq. (2c).