

4

Linked-List Data Structures

In this chapter we shall study:

- singly-linked lists and doubly-linked lists (Section 4.1)
- linked-list insertion, deletion, searching, merging, and sorting algorithms (Sections 4.2–6).

This chapter interleaves its explanations of singly-linked lists and doubly-linked lists. If you prefer, on a first reading you can skip the material on doubly-linked lists, namely Sections 4.1.2, 4.2.2, and 4.3.2.

4.1 Linked lists

A *linked list* consists of a sequence of *nodes*, connected by *links*. Each node contains a single element, together with links to one or both neighboring nodes. Figures 4.2 and 4.6 show some examples of linked lists.

A node's *successor* is the next node in the sequence, and its *predecessor* is the previous node in the sequence. The last node in the sequence has no successor, and the first node in the sequence has no predecessor.

By convention, we use a special *null link* wherever there is no node to link to. Thus every node in a linked list contains a link to its successor, but the last node contains a null link instead (indicating that there is no successor).

We routinely use diagrams such as Figures 4.2 and 4.6 to illustrate linked lists. A link is shown as a small black circle at the tail of an arrow, and a null link is shown as a small black circle without an arrow. A node is shown as a box, which contains an element and one or more links. These conventions are summarized in Figure 4.1.

The *length* of a linked list is the number of nodes (elements) in it. A linked list is *empty* if it has length zero, i.e., no nodes at all.

A linked list can be of any length, and can therefore contain any number of elements. We can access the element in any node, provided that we have a link to that node.

Another important property of linked lists is that we can manipulate the links. This makes it possible to achieve a variety of effects. We can insert and delete nodes, thus