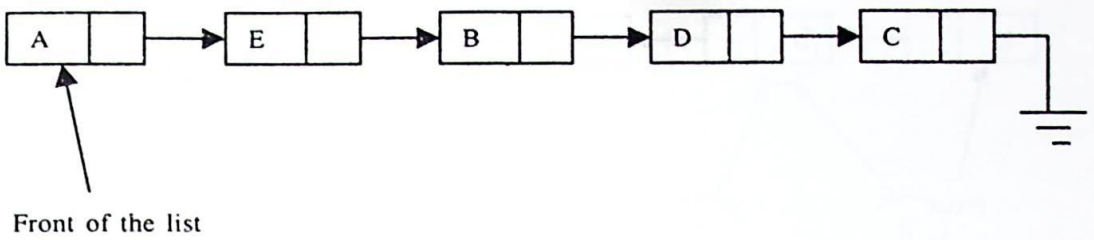

FIGURE 1.3 Linked list.

1.8.3 Stack

A stack is a list with the restriction that insertions and deletions can be performed at one end of the list. The fundamental operations on a stack are PUSH and POP. PUSH inserts an element at the designated end and POP deletes an element from the same end. Applying POP on an empty stack and applying PUSH to a full stack (implementation dependent) leads to an error situation. Stack is also called LIFO (last in first out list). A stack can be implemented using a linked list or an array. Figure 1.4 shows the effect of push and pop operations on a stack.

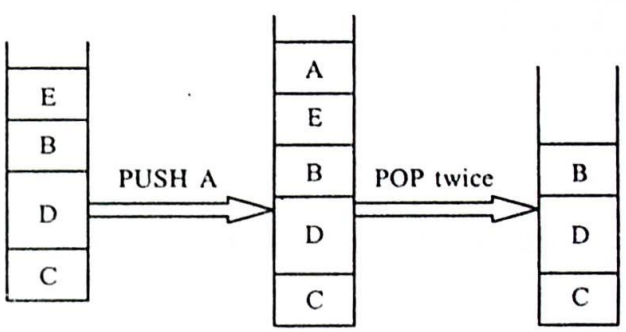

FIGURE 1.4 Stack.

1.8.4 Queue

Like stacks, queues are lists but have insertion done at one end and deletion at the other end. Queues follow the 'first in first out' (FIFO) discipline. The basic operations on a queue are 'enque' which inserts an element at the end of the list (the rear end) and 'dequeue' which deletes the element at the start of the list (the front of the queue). Figure 1.5(a) schematically shows a queue.

After insertion of element A, the queue becomes as shown in Figure 1.5(b) and after deletion of element E, it becomes as shown in Figure 1.5(c).

Queues may also be implemented using arrays. Dequeing from an empty queue leads to error. So also, enqueing in a full queue (implementation limitation) leads to implementation error.

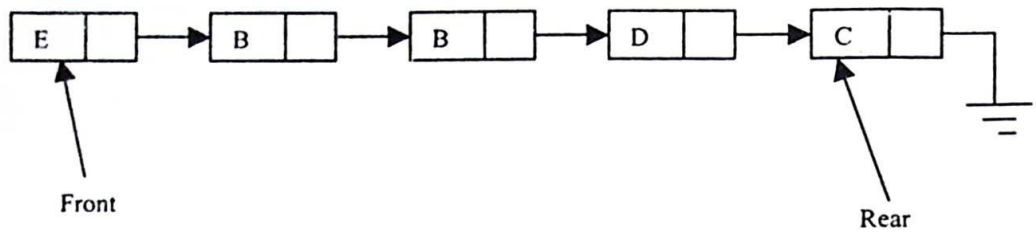

FIGURE 1.5(a) A queue.

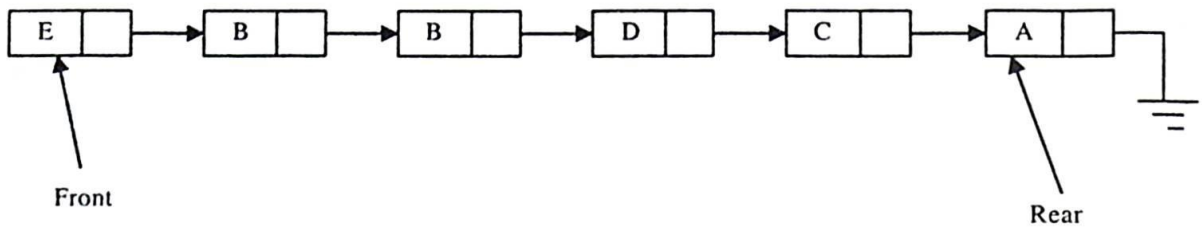

FIGURE 1.5(b) Insertion of an element.

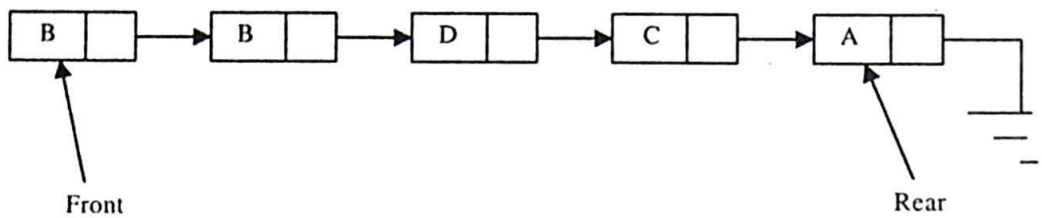

FIGURE 1.5(c) Deletion of an element.

1.8.5 Set

A set is a collection of non-repetitive elements where each element of a set is either a set or atomic. The atoms are usually integers, characters, strings, etc. and all elements in a set are usually of the same type. The most common operations on sets are union, intersection, difference, membership checking, insertion of a new element, deletion of an element, finding the extremal element, checking for equality of two sets, etc.

Sets can be implemented using linked lists, bit vectors, various kinds of trees, such as binary search trees, tries and balanced trees. We show how to represent disjoint sets by using trees. Suppose, we have two sets $S_1 = \{1, 3, 5, 9\}$ and $S_2 = \{6, 7, 8\}$. Each set has a representative that is a member of the set. For example, in Figure 1.6(a), 1 is a representative of S_1 and 6 is a representative of S_2.

Two important operations on sets are FIND and UNION. FIND(8) will give us 6 (the representative). The UNION of S_1 and S_2 will give us a set as represented in Figure 1.6(b).

A simple way to implement a disjoint-set structure is to represent each set by a linked list. The first object in each linked list serves as the set's representative. Each object in the linked list contains a set member, a pointer to the node containing the next member of the set, and a pointer back to the representative. We have two other pointers: one to the head

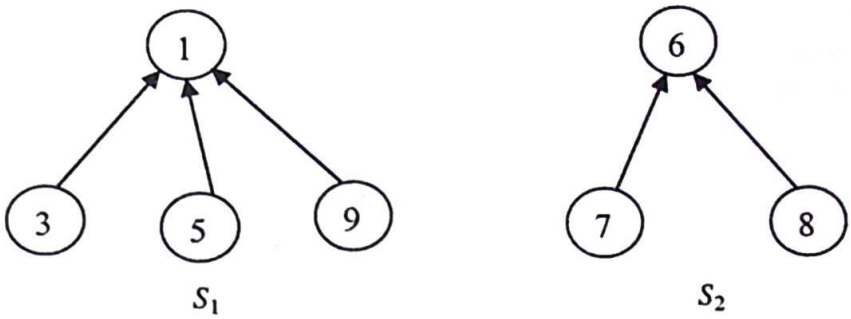

FIGURE 1.6(a) Disjoint sets.

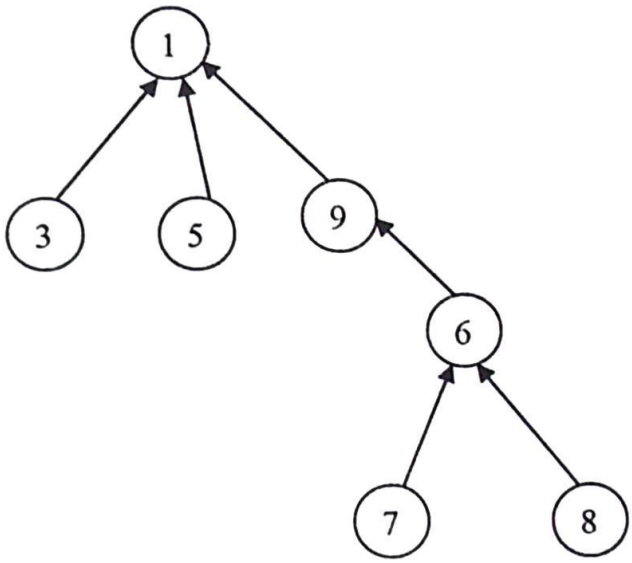

FIGURE 1.6(b) Set union.

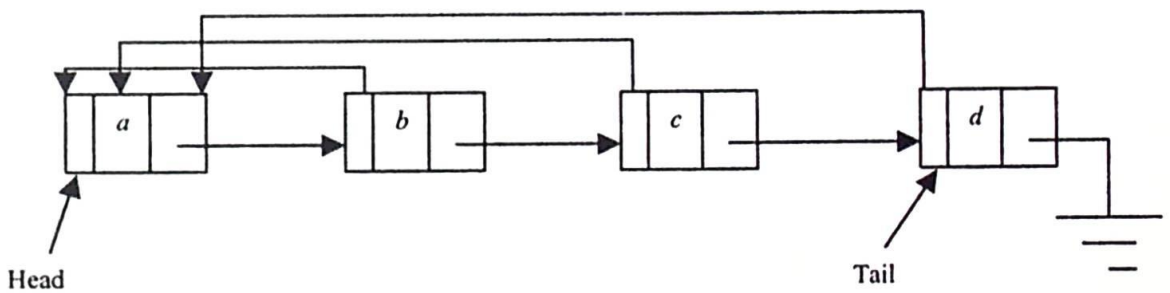

FIGURE 1.6(c) Set represented as a linked list.

of the list (first object) another to the tail (last object) of the list. Thus, the linked list representation for S, where $S = \{a, b, c, d\}$ will be as shown in Figure 1.6(c).

An alternative to the list representation is the bit-vector representation of sets. Let us assume that the universal set has n linearly ordered members. A subset S is represented as

a vector of n bits, where the i^{th} bit is 1 if the i^{th} element of the universal set is a member of S. The bit-vector representation has the advantage that one can determine whether the i^{th} element of the universal set is a member of a set in time independent of the size of the set. The basic operations on sets such as union and intersection can be carried out through the bit operations OR, AND.

A set can also be represented by an array A such that $A(i) = 1$ if and only if the i^{th} member of the universal set is in S. With an array representation, it is easy to determine whether an element is a member of a set. The disadvantage is that union and intersection require time proportional to n rather than the sizes of the sets involved. The space required to store S is proportional to n.

1.8.6 Graph

A graph $G = (V, E)$ is a set of vertices V and a set of edges E, where $E \subseteq V \times V$. A graph may be represented in two ways: (i) adjacency list (ii) adjacency matrix. Two vertices are adjacent if there is an edge between them. In the case of adjacency list, we maintain, as many lists as there are vertices in the graph. For each vertex of the graph, we maintain a list containing the vertices that are adjacent to the vertex.

For the graph in Figure 1.7(a), the adjacency lists are shown in Figure 1.7(b).

The alternative representation uses a matrix. The $(i, j)^{th}$ element of the matrix is 1 if there is an edge from vertex i to vertex j, and is 0 otherwise. For the graph in Figure 1.7(a), the adjacency matrix is shown in Figure 1.7(c).

For the undirected graph, the adjacency matrix is symmetric.

1.8.7 Tree

A tree is a connected graph without cycle. There are many alternative definitions of a tree. For example, a tree may be defined as a graph where there is exactly one path between any pair of vertices. We are more interested in rooted binary trees. A rooted binary tree has a distinguished vertex called the root (A in Figure 1.8(b)) of the tree and if the tree is also

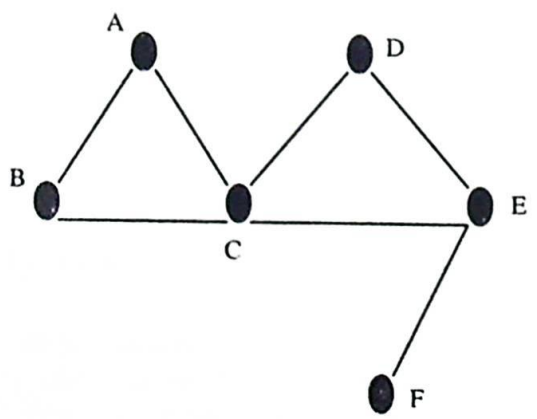

FIGURE 1.7(a) A sample graph.