```
    if (itemA < itemB)
        GetNextItem(itemA) from self
    else if itemB < itemA
        setB.GetNextItem(itemB)
    else
        result.Store(itemA)
        GetNextItem(itemA) from self
        setB.GetNextItem(itemB)
return result
```

Be very careful: Sets are *unordered* collections, but the implementation structure that contains the sets can be ordered—and we can take advantage of that ordering to produce an O(N) algorithm for the intersection operation. This process of moving down two ordered lists in parallel is called a *merge*. This very useful algorithm can be used to implement the other binary operations in O(N) time. We leave the design of these algorithms as exercises.

Both implementations are left as programming assignments.

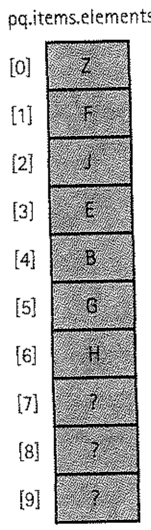
Summary

In this chapter, we discussed several branching structures: trees, heaps, and graphs. Branching structures are very versatile and offer a good way to model many real-world objects and situations. Because these data structures are appropriate for many different types of applications, all kinds of variations and generalizations of trees and graphs exist. These topics are introduced here to highlight the wide variety of applications for which programmers must select and create appropriate data structures. They are generally covered in detail in more advanced computer science courses.

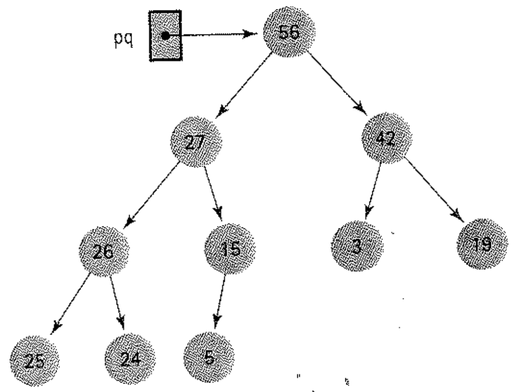
We also developed the Set ADT. Two types of implementation structures are possible for this ADT: explicit, in which each item of the base type is associated with a Boolean flag, and implicit, in which the items in the set are kept in a list.

Exercises

1. A priority queue containing characters is implemented as a heap stored in an array. The precondition states that this priority queue cannot contain duplicate elements. Currently, the priority queue holds 10 elements, as shown on the next page. What values might be stored in array positions 7–9 so that the properties of a heap are satisfied?



2. A *minimum heap* has the following order property: The value of each element is less than or equal to the value of each of its children. What changes must be made in the heap operations given in this chapter?
3. a. Write a nonrecursive version of `ReheapDown`.
   b. Write a nonrecursive version of `ReheapUp`.
   c. Describe the nonrecursive versions of these operations in terms of Big-O notation.
4. A priority queue is implemented as a heap:



   a. Show how the heap would look after this series of operations:

```
pq.Enqueue(28);
pq.Enqueue(2);
pq.Enqueue(40);
pq.Dequeue(x);
pq.Dequeue(y);
pq.Dequeue(z);
```

   b. What would the values of `x`, `y`, and `z` be after the series of operations in part (a)?

5. A priority queue is implemented as a linked list, sorted from largest to smallest element.
   a. How would the definition of `PQType` change?
   b. Write the `Enqueue` operation, using this implementation.
   c. Write the `Dequeue` operation, using this implementation.
   d. Compare the `Enqueue` and `Dequeue` operations to those for the heap implementation, in terms of Big-O notation.
6. A priority queue is implemented as a binary search tree.
   a. How would the definition of `PQType` change?
   b. Write the `Enqueue` operation, using this implementation.
   c. Write the `Dequeue` operation, using this implementation.
   d. Compare the `Enqueue` and `Dequeue` operations to those for the heap implementation, in terms of Big-O notation. Under what conditions would this implementation be better or worse than the heap implementation?
7. A priority queue is implemented as a sequential array-based list. The highest-priority item is in the first array position, the second-highest priority item is in the second array position, and so on.
   a. Write the declarations in the private part of the priority queue class definition needed for this implementation.
   b. Write the `Enqueue` operation, using this implementation.
   c. Write the `Dequeue` operation, using this implementation.
   d. Compare the `Enqueue` and `Dequeue` operations to those for the heap implementation, in terms of Big-O notation. Under what conditions would this implementation be better or worse than the heap implementation?
8. A stack is implemented using a priority queue. Each element is time-stamped as it is put into the stack. (The time stamp is a number between 0 and `INT_MAX`. Each time an element is pushed onto the stack, it is assigned the next larger number.)
   a. What is the highest-priority element?
   b. Write the `Push` and `Pop` algorithms, using the specifications in Chapter 4.
   c. Compare these `Push` and `Pop` operations to the ones implemented in Chapter 4, in terms of Big-O notation.
9. A FIFO queue is implemented using a priority queue. Each element is time-stamped as it is put into the queue. (The time stamp is a number between 0 and

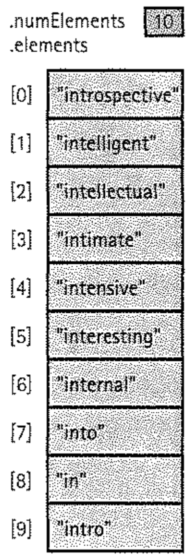
INT_MAX. Each time an element is enqueued, it is assigned the next larger number.)

a. What is the highest-priority element?

b. Write the Enqueue and Dequeue operations, using the specifications in Chapter 4.

c. Compare these Enqueue and Dequeue operations to the ones implemented in Chapter 4, in terms of Big-O notation.

10. A priority queue of strings is implemented using a heap. The heap contains the following elements:



a. What feature of these strings is used to determine their priority in the priority queue?

b. Show how this priority queue is affected by adding the string "interviewing."

Use the following description of an *undirected graph* for Exercises 11–14.

EmployeeGraph	=	(V, E)
V(EmployeeGraph)	=	{Susan, Darlene, Mike, Fred, John, Sander, Lance, Jean, Brent, Fran}
E(EmployeeGraph)	=	{(Susan, Darlene), (Fred, Brent), (Sander, Susan), (Lance, Fran), (Sander, Fran), (Fran, John), (Lance, Jean), (Jean, Susan), (Mike, Darlene), (Brent, Lance), (Susan, John)}

11. Draw a picture of EmployeeGraph.

12. Draw `EmployeeGraph`, implemented as an adjacency matrix. Store the vertex values in alphabetical order.
13. Using the adjacency matrix for `EmployeeGraph` from Exercise 12, describe the path from Susan to Lance
    a. using a breadth-first strategy.
    b. using a depth-first strategy.
14. Which one of the following phrases best describes the relationship represented by the edges between the vertices in `EmployeeGraph`?
    a. "works for"
    b. "is the supervisor of"
    c. "is senior to"
    d. "works with"

Use the following specification of a *directed graph* for Exercises 15–18.

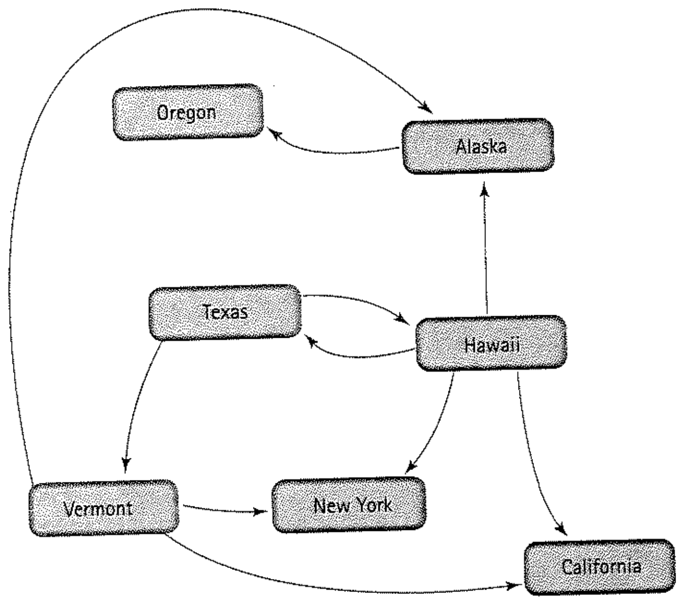
ZooGraph = (V, E)

V(ZooGraph) = {dog, cat, animal, vertebrate, oyster, shellfish, invertebrate, crab, poodle, monkey, banana, dalmatian, dachshund}

E(ZooGraph) = {(vertebrate, animal), (invertebrate, animal), (dog, vertebrate), (cat, vertebrate), (monkey, vertebrate), (shellfish, invertebrate), (crab, shellfish), (oyster, shellfish), (poodle, dog), (dalmatian, dog), (dachshund, dog)}

15. Draw a picture of `ZooGraph`.
16. Draw the adjacency matrix for `ZooGraph`. Store the vertices in alphabetical order.
17. To tell if one element in `ZooGraph` has relation X to another element, you look for a path between them. Show whether the following statements are true, using the picture or adjacency matrix.
    a. dalmatian X dog
    b. dalmatian X vertebrate
    c. dalmatian X poodle
    d. banana X invertebrate
    e. oyster X invertebrate
    f. monkey X invertebrate
18. Which of the following phrases best describes relation X in Exercise 17?
    a. "has a"
    b. "is an example of"
    c. "is a generalization of"
    d. "eats"

Use the following graph for Exercises 19–21.



19. Describe the graph pictured above, using the formal graph notation.

    V(StateGraph) =

    E(StateGraph) =

20. a. Is there a path from Oregon to any other state in the graph?

    b. Is there a path from Hawaii to every other state in the graph?

    c. From which state(s) in the graph is there a path to Hawaii?

21. a. Show the adjacency matrix that would describe the edges in the graph. Store the vertices in alphabetical order.

    b. Show the array-of-pointers adjacency lists that would describe the edges in the graph.

22. Extend the class `GraphType` in this chapter to include a Boolean `EdgeExists` operation, which determines whether two vertices are connected by an edge.

    a. Write the declaration of this function. Include adequate comments.

    b. Using the adjacency matrix implementation developed in the chapter and the declaration from part (a), implement the body of the function.

23. Extend the class `GraphType` in this chapter to include a `DeleteEdge` operation, which deletes a given edge.

    a. Write the declaration of this function. Include adequate comments.

b. Using the adjacency matrix implementation developed in the chapter and the declaration from part (a), implement the body of the function.

24. Extend the class `GraphType` in this chapter to include a `DeleteVertex` operation, which deletes a vertex from the graph. Deleting a vertex is more complicated than deleting an edge from the graph. Discuss why.

25. The `DepthFirstSearch` operation can be implemented without a stack by using recursion.
    a. Name the base case(s). Name the general case(s).
    b. Write the algorithm for a recursive depth-first search.

26. Distinguish between set representations that are implicit and those that are explicit.

27. Finish designing the algorithms for the explicit set representation.

28. Finish designing the algorithms for the implicit set representation.

29. Implement the copy constructor for PQType.

30. Did you notice that we did not include a copy constructor from GraphType? Discuss the issues involved in implementing this copy constructor.

ph. Store

edges in

eExists
:dge.

:r and the

)peration,