3.6.2 Binary search trees

A binary search tree is a binary tree in which the vertices are labeled with elements from a linearly ordered set in such a way that all elements stored in the left subtree of a vertex \( v \) are less than the element stored at vertex \( v \), and all elements stored in the right subtree of a vertex \( v \) are greater than the element stored at vertex \( v \). This condition, which is called the binary search tree property, holds for every vertex of a binary search tree. The representation of a set by a binary search tree is not unique; in the worst case it may be a degenerate tree, i.e., a tree in which each internal vertex has exactly one child. Figure 3.9 shows two binary search trees representing the same set.

![Binary Search Trees](image)

Fig. 3.9 Two binary search trees representing the same set.

The operations supported by this data structure are insertion, deletion, testing for membership and retrieving the minimum or maximum.

3.7 Exercises

3.1. Write an algorithm to delete an element \( x \), if it exists, from a doubly-linked list \( L \). Assume that the variable head points to the first element in the list and the functions pred(\( y \)) and next(\( y \)) return the predecessor and successor of node \( y \), respectively.

3.2. Give an algorithm to test whether a list has a repeated element.

3.3. Rewrite Algorithm insertionSort so that its input is a doubly linked list of \( n \) elements instead of an array. Will the time complexity change? Is the new algorithm more efficient?

3.4. A polynomial of the form \( p(x) = a_1x^{b_1} + a_2x^{b_2} + \ldots + a_nx^{b_n} \), where \( b_1 > b_2 > \ldots > b_n \geq 0 \), can be represented by a linked list in which each record has three fields for \( a_i \), \( b_i \) and the link to the next record. Give an algorithm to add two polynomials using this representation. What is the running time of your algorithm?
3.5. Give the adjacency matrix and adjacency list representations of the graph shown in Fig. 3.5.

3.6. Describe an algorithm to insert and delete edges in the adjacency list representation for
(a) a directed graph.
(b) an undirected graph.

3.7. Let $S_1$ be a stack containing $n$ elements. Give an algorithm to sort the elements in $S_1$ so that the smallest element is on top of the stack after sorting. Assume you are allowed to use another stack $S_2$ as a temporary storage. What is the time complexity of your algorithm?

3.8. What if you are allowed to use two stacks $S_2$ and $S_3$ as a temporary storage in Exercise 3.7?

3.9. Let $G$ be a directed graph with $n$ vertices and $m$ edges. When is it the case that the adjacency matrix representation is more efficient than the adjacency lists representation? Explain.

3.10. Prove that a graph is bipartite if and only if it has no odd-length cycles.

3.11. Draw the almost-complete binary tree with
(a) 10 nodes.
(b) 19 nodes.


3.13. Prove Observation 3.2.


3.15. Prove Observation 3.3.

3.16. Prove Observation 3.5.

3.17. Is a tree a bipartite graph? Prove your answer (see Exercise 3.10).

3.18. Let $T$ be a nonempty binary search tree. Give an algorithm to
(a) return the minimum element stored in $T$.
(b) return the maximum element stored in $T$.

3.19. Let $T$ be a nonempty binary search tree. Give an algorithm to list all the elements in $T$ in increasing order. What is the time complexity of your algorithm?

3.20. Let $T$ be a nonempty binary search tree. Give an algorithm to delete an element $x$ from $T$, if it exists. What is the time complexity of your algorithm?

3.21. Let $T$ be binary search tree. Give an algorithm to insert an element $x$ in its proper position in $T$. What is the time complexity of your algorithm?
3.22. What is the time complexity of deletion and insertion in a binary search tree? Explain.

3.23. When discussing the time complexity of an operation in a binary search tree, which of the $O$ and $\Theta$ notations is more appropriate? Explain.

3.8 Bibliographic notes

This chapter outlines some of the basic data structures that are frequently used in the design and analysis of algorithms. More detailed treatment can be found in many books on data structures. These include, among others, Aho, Hopcroft and Ullman (1983), Gonnet (1984), Knuth (1968), Knuth (1973), Reingold and Hansen (1983), Standish (1980), Tarjan (1983) and Wirth (1986). The definitions in this chapter conform with those in Tarjan (1983). The adjacency lists data structure was suggested by Tarjan and is described in Tarjan (1972) and Hopcroft and Tarjan (1973).