Trees

Chapter 4
Objectives

- Understand the terminology of the tree data structure
- Represent a tree structure in a program
- Understand the importance of the binary trees
- Use a binary search tree for storing ordered elements
Motivation

- Why lists, stacks, and queues are not enough?
- Not everything can be linearized. We may need to represent hierarchies, for example.
- Sorted array search: $O(\log(n))$
- Sorted array insert: $O(n)$
- Linked list search: $O(n)$
- Linked list insert: $O(1)$
- Can we build a data structure that is fast for both search and insert?
Hierarchical Structures

UC System

UCR

BCOE

CSE

UCI

CNAS

ECE

UCSD

…
Hierarchical Structures
Definition

- A tree can be defined recursively
- A tree is a group of nodes
- Each node contains a value
- If the tree is not empty, one node is identified as the **root node**
- The root node has zero or more **subtrees**
- The root of a subtree is connected to the root of the tree
**Terminology: Basic Definitions**

A is the parent of D
D is the child of A

B, C, and D are siblings

E and F are **not** siblings

Subtrees
Terminology: Descendants

Descendants of A
Terminology: Ancestors

Ancestors of E:
- A
- B
- C

Descendant of E:
- E
- I
- J
Terminology: Leaves

Internal nodes

Leaf nodes (Leaves)
Terminology: Levels, Depth

Level 0
A

Level 1
B
C
D

Level 2
E
F
G
H

Level 3
I
J
K

What is the height of the tree?

J is at level 3
The depth of J is 3

What is the relationship between the depth of a node and the number of ancestors?
The path from A to J is (A, B, E, J)
The length of the path is three (edges)

What is the path from D to K?
Tree Representation

```
template <type T>
class Tree {
    class Node {
        T value;
        list<Node*> children;
    };
    Node* root;
};
```
Parent Representation

```cpp
template <typename T>
class Tree {
    class Node {
        T value;
        Node* parent;
    };
    list<Node*> nodes;
};
```
Left-child Right-sibling
Left-child Right-sibling

```
template <typename T>
class Tree {
    class Node {
    T value;
    Node* left_child;
    Node* right_sibling;
    }
    Node* root;
};
```
Binary Trees

- A special case where every node has at most two children
- Has many applications that make it particularly interesting
- More restricted → Room for optimization

```cpp
template <typename T>
class Tree {
    class Node {
        T value;
        Node* left;
        Node* right;
    };
    Node* root;
};
```
Application: Expression Tree

(3 × 5 + 4/2) × 2
Inorder Tree Traversal

$$((3 \times 5) + (4/2)) \times 2$$
Postorder Tree Traversal

35 × 42/+2 ×
Preorder Tree Traversal

\(\times + \times \frac{35}{422}\)
Implementation of Traversals

```c
inorder(Node* root) {
    if (root == null)
        return;
    inorder(root->left);
    print(root->value);
    inorder(root->right);
}
```

```c
postorder(Node* root) {
    if (root == null)
        return;
    postorder(root->left);
    postorder(root->right);
    print(root->value);
}
```

```c
preorder(Node* root) {
    if (root == null)
        return;
    print(root->value);
    preorder(root->left);
    preorder(root->right);
}
```