2. Tree Concept

2.1 Basic Theory

2.1.1 Definition of a Tree

image

A **Tree** is a **non-linear hierarchical** data structure composed of nodes and edges.

- Every Tree has a **root** (root node).
- The root can have **children**.
- A node with no children is called a **leaf**.
- Each node and its descendants can form a **subtree**.

2.1.2 Important Terms in a Tree

image

- **Root** → the topmost node.
- **Parent** → a node that has children.
- **Child** → a node derived from a parent.
- **Leaf** → a node without children.
- **Subtree** → a small tree formed from a node and its descendants.

2.1.3 Binary Tree

image

A **Binary Tree** is a tree data structure in which each node has at most two children (left and right).

- **There are no special rules** for placing node values.
- **Used** for hierarchical representation, data structures like heaps, expression trees, and implementing search or traversal algorithms.
- Nodes in a Binary Tree can have 0, 1, or 2 children.

Characteristics of a Binary Tree:

- Height of the tree: the number of levels from the root to the farthest leaf.

- Leaf node: a node that has no children.

2.2 Introduction to Stack and Queue

2.2.1 Stack

image

A LIFO (Last In First Out) data structure → the last data in is the first one out. Example: a stack of plates in a cafeteria. The last plate placed on top will be the first one taken.

Main operations:

- push(x) → adds data to the top of the stack.
- pop() → removes data from the top of the stack.

Application in Trees Used in DFS (Depth First Search) or Preorder, Inorder, Postorder traversals. When we enter a child node, the current node is stored on the stack. After finishing with the child, we pop from the stack to continue.

2.2.2 Queue

image

A FIFO (First In First Out) data structure → the first data in is the first one out. Example: a minimarket cashier line, the person who arrives first will be served first.

Main operations:

- enqueue(x) → inserts data at the back of the queue.
- dequeue() → removes data from the front of the queue.

Application in Trees Used in BFS (Breadth First Search) or Level Order traversal. When visiting a node, its children are added to the queue, then processed one by one in order.

2.3 Traversal in Trees

Traversal is the process of visiting each node in a tree.

2.3.1 Inorder (Left, Root, Right)

- visit the left, then the middle (root), then the right. The result is usually data sorted in ascending order.

```
// Inorder (Left - Root - Right)
void inorder(Node* root) {
   if (root == nullptr) return;
   inorder(root->left);
   cout << root->data << " ";
   inorder(root->right);
}
```

2.3.2 Preorder (Root, Left, Right)

- visit the middle first, then the left, then the right. Used to represent the structure of the tree.

```
// Preorder (Root - Left - Right)
void preorder(Node* root) {
   if (root == nullptr) return;
   cout << root->data << " ";
   preorder(root->left);
   preorder(root->right);
}
```

2.3.3 Postorder (Left, Right, Root)

- visit the left first, then the right, finally the middle. Suitable if you want to delete all contents of the tree.

```
// Postorder (Left - Right - Root)
void postorder(Node* root) {
   if (root == nullptr) return;
   postorder(root->left);
   postorder(root->right);
   cout << root->data << " ";
}
```

2.3.4 Level Order

- Visit nodes level by level from top to bottom, left to right.
- Usually uses a queue.

```
void levelOrder(Node* root) {
   if (root == nullptr) return;
   queue<Node*> q;
   q.push(root);
   while (!q.empty()) {
   Node* temp = q.front();
   q.pop();
   cout << temp->data << " ";
   if (temp->left) q.push(temp->left);
   if (temp->right) q.push(temp->right);
   }
}
```

2.3.5 Traversal Illustration Example

```
     50
    /  \
   /    \
  30     70
 / \    / \
20 40  60 80
```

- **Inorder:** 20, 30, 40, 50, 60, 70, 80
- **Preorder:** 50, 30, 20, 40, 70, 60, 80
- **Postorder:** 20, 40, 30, 60, 80, 70, 50
- **Level Order:** 50, 30, 70, 20, 40, 60, 80

References

- GeeksforGeeks, “Binary Search Tree (BST),” [Online]. Available: https://www.geeksforgeeks.org/binary-search-tree-data-structure/. [Accessed: 20-August-2025]
- Programiz, “Tree Data Structure in C++,” [Online]. Available: https://www.programiz.com/dsa/binary-search-tree. [Accessed: 20-August-2025]

Revision #7
Created 2025-09-06 10:41:02 UTC by YP
Updated 2025-10-30 00:34:05 UTC by RE