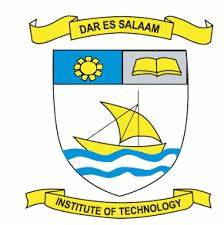
**DAR ES SALAAM INSTITUTE OF TECHNOLOGY**



**CLASS:**

**OD22 IT**

**FUNDAMENTAL OF DATA STRUCTURE AND ALGORITHM**

**ITT 05213**

**YUSTIN MWINUKA**

**14TH MAY 2024.**

**MODULE NAME:**

**MODULE CODE:**

**LECTURE:**

**DATE:**

**GROUP ASSIGNMENT**

|  |  |  |
| --- | --- | --- |
| **SN** | **NAME** | **REG.NO** |
| **1** | **NOEL MOSES SAMWEL** | **220229416717** |
| **2** | **MASTOKA JUMANNE MASTOKA** | **220229375291** |
| **3** | **SULEIMAN MANSOUR ALI** | **220229419885** |

Tree and graph data structures are fundamental concepts in computer science, used extensively in various applications such as databases, networking, AI, and more. Both structures represent a collection of interconnected elements, but they differ significantly in their properties, usage, and operations.

**Tree Data Structures**

A tree is a hierarchical data structure with a set of connected nodes. Each node stores a value or data and references to its child nodes. The structure starts from a root node and expands outward, forming a tree-like shape.

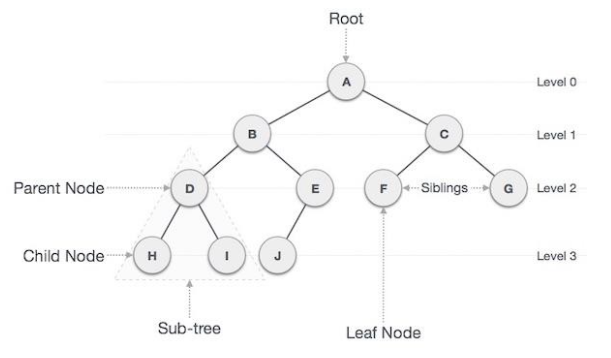
**Characteristics of Trees:**

1. Hierarchical Structure: Nodes are arranged in levels, with the root at the top level.

2. Single Path: There is exactly one path between any two nodes.

3. Parent-Child Relationship: Each node (except the root) has exactly one parent and can have zero or more children.

4. Acyclic: Trees do not contain cycles.



Example of tree data structure

**Types of Trees:**

1. Binary Tree: Each node has at most two children, referred to as the left child and the right child.

2. Binary Search Tree (BST): A binary tree with the property that for any node, the left subtree contains only nodes with values less than the node’s value, and the right subtree contains only nodes with values greater than the node’s value.

3. Balanced Trees: Trees like AVL trees and Red-Black trees that maintain a balanced height to ensure operations like insertion, deletion, and lookup are efficient.

4. Heap: A binary tree where the parent node is always greater than or equal to (max-heap) or less than or equal to (min-heap) its children.

5. B-Tree: A self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. Commonly used in databases and filesystems.

**Operations on Trees:**

1. Insertion: Adding a node to the tree while maintaining its properties.

2. Deletion: Removing a node and reconfiguring the tree to preserve its properties.

3. Traversal: Visiting all the nodes in the tree in a specific order. Common traversal methods include:

- In-order (Left, Root, Right): Often used in BSTs to retrieve sorted order.

- Pre-order (Root, Left, Right): Used to create a copy of the tree.

- Post-order (Left, Right, Root): Used to delete the tree.

- Level-order: Visiting nodes level by level from top to bottom.

**Graph Data Structures**

A graph is a more generalized data structure consisting of a set of nodes (vertices) and a set of edges that connect pairs of nodes. Unlike trees, graphs can represent more complex relationships.

**Characteristics of Graphs:**

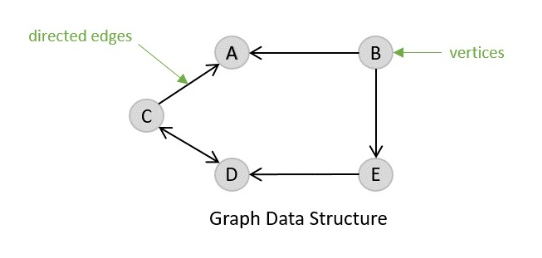
1. Nodes (Vertices): Fundamental units of the graph.

2. Edges: Connections between nodes, which can be directed or undirected.

3. Cycles: Graphs can contain cycles, where a path of edges can form a loop.

4. Paths: Multiple paths can exist between nodes.

5. Weights: Edges can have weights, representing the cost or distance between nodes.



Examples of graph data structure

**Types of Graphs:**

1. Undirected Graph: Edges have no direction, meaning the relationship between nodes is bidirectional.

2. Directed Graph (Digraph): Edges have a direction, indicating the relationship is one-way.

3. Weighted Graph: Edges have associated weights.

4. Unweighted Graph: Edges do not have weights.

5. Cyclic Graph: Contains at least one cycle.

6. Acyclic Graph: Does not contain any cycles. A Directed Acyclic Graph (DAG) is particularly important in scenarios like task scheduling.

**Operations on Graphs:**

1. Insertion: Adding a vertex or edge.

2. Deletion: Removing a vertex or edge.

3. Traversal: Visiting all nodes in the graph. Common traversal methods include:

Depth-First Search (DFS): Explores as far as possible along each branch before backtracking.

Breadth-First Search (BFS): Explores all neighbours at the present depth level before moving on to nodes at the next depth level.

4. Shortest Path: Finding the shortest path between nodes (e.g., Dijkstra’s algorithm, Bellman-Ford algorithm).

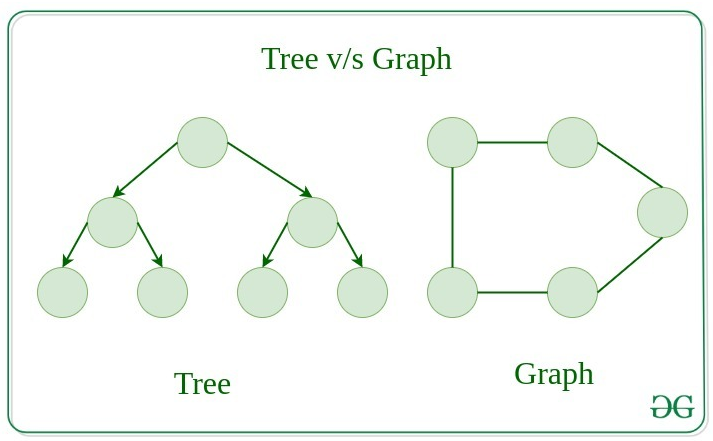
5. Minimum Spanning Tree (MST): Finding the subset of edges that connect all vertices with the minimum total edge weight (e.g., Kruskal’s algorithm, Prim’s algorithm).

**Comparison between Trees and Graphs:**

Structure: Trees are a special type of graph with hierarchical structure and no cycles, while graphs are more general and can have cycles.

Traversal: Trees have a natural hierarchical traversal, whereas graph traversal depends on specific algorithms like DFS and BFS.

Usage: Trees are used in scenarios requiring hierarchical data representation (e.g., file systems, organizational structures), while graphs are used for representing more complex networks (e.g., social networks, transportation networks).



**Applications:**

Trees:

1. Binary Search Trees (BST): Efficient searching, insertion, and deletion operations.

2. Heaps: Priority queues in algorithms like Dijkstra’s shortest path.

3. B-Trees: Databases and file systems for efficient data retrieval.

Graphs:

1. Social Networks: Modeling relationships and connections.

2. Internet Routing: Representing and optimizing network paths.

3. AI and Machine Learning: Representing states and transitions in search problems.

Both trees and graphs are versatile and powerful data structures that play crucial roles in computer science and various applications. Understanding their properties, types, and operations is essential for designing efficient algorithms and solving complex problems.