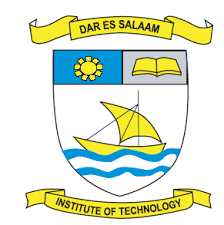
DAR ES SALAAM INSTITUTE OF TECHNOLOGY (DIT).



DEPARTMENT: COMPUTER STUDIES DEPARTMENT.

MODULE: FUNDAMENTALS OF DATA STRUCTURE AND ALGORITHM.

MODULE CODE: ITT 05217.

CLASS: OD22-IT.

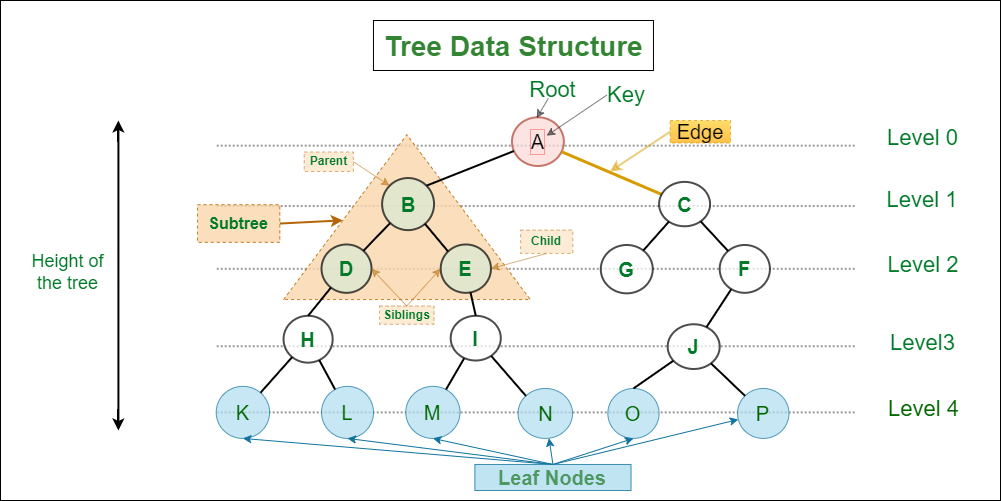
ASSIGNMENT: 02.

GROUP MEMBERS:

|  |  |  |
| --- | --- | --- |
| S/N | NAMES | REGISTRATION NUMBER |
| 1. | ALEX SALVATORY MSELE | 220229363851 |
| 2. | IRENE PETRO BIRRO | 220229323424 |
| 3. | AISHA ABDULRAHIM ABDUL-AZIZ | 220229374492 |

**TREE DATA STRUCTURE**

A tree is a specialized data structure used to store data in a hierarchical manner. It is commonly used to organize and represent data in a way that is easy to navigate and search.



Here are some key points about tree data structures:

**1. Hierarchy and Relationships:**

- A tree consists of a collection of nodes connected by **edges**.

- The topmost node in a tree is called the **root**.

- Each node can have multiple child nodes, forming a hierarchical relationship.

- Nodes below a parent node are called its **children**.

- Nodes with no children are called **leaf nodes** or **external nodes**.

- Nodes with at least one child are called **internal nodes**.

- Nodes can also have **ancestors** (predecessor nodes on the path to the root) and **descendants** (nodes reachable from a given node).

**2. Basic TerminologIies:**

- Parent Node: The predecessor of a node is called its parent.

- Child Node: The immediate successor of a node is its child.

- Root Node: The topmost node with no parent.

- Leaf Node: Nodes with no children.

- Ancestor: Predecessor nodes on the path to a given node.

- Descendant: A node reachable from another node.

- Sibling: Children of the same parent.

- Level of a Node: The count of edges on the path from the root to that node.

**3. Non-Linear Structure:**

- Unlike arrays or lists, tree data is **not stored linearly**.

- Trees **are arranged in multiple levels**, forming a hierarchical structure.

-For these reasons, trees are considered non-linear data structures.

**4. Types of Trees:**

**- Binary Tree:** Each node has at most two children.

- **Binary Search Tree (BST):** A binary tree where the left child is less than or equal to the parent, and the right child is greater.

- **AVL Tree:** A self-balancing binary search tree.

- **B-tree:** A balanced tree used for databases and file systems.

- Many other specialized trees exist, such as **trie, segment tree,** and **heap.**

**5. Applications:**

- Trees are used in various domains, including:

* Representing hierarchical data (e.g., file systems, organization charts).
* Implementing search algorithms (e.g., binary search in BST).
* Parsing expressions (e.g., expression trees).
* Decision-making (e.g., decision trees in machine learning).

**6. Advantages and Disadvantages:**

- Advantages:

* Efficient for hierarchical data.
* Fast search, insertion, and deletion (in balanced trees).

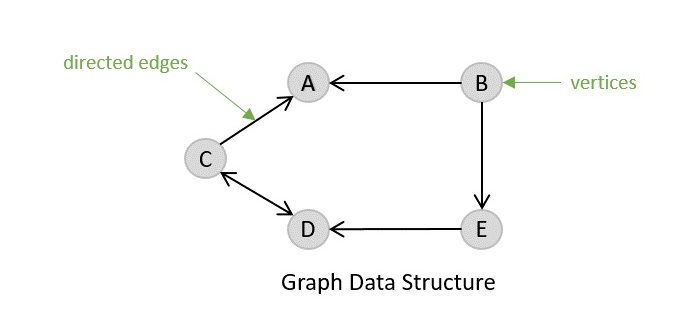
- Disadvantages:

* Requires memory for pointers/edges.
* Complex to implement and maintain.

In summary, **a tree data structure** **provides an organized and efficient way to represent and manage hierarchical data**. It is widely used in computer science and various applications.

**GRAPH DATA STRUCTURE**

A graph data structure is a collection of nodes (also known as vertices) connected by edges. It's used to represent relationships between different entities.



Here are some key points about graph data structures:

1. **Terminologies:**

a. Vertices (Nodes):

- Vertices are the fundamental units of the graph.

- They represent individual entities or objects.

- Each vertex can be labeled or unlabeled.

b. Edges:

- Edges connect two nodes in the graph.

- They represent relationships or connections between vertices.

- Edges can be ordered pairs of nodes in a directed graph.

- There are no strict rules for how edges connect nodes.

**2. Graph Types:**

Graphs can be categorized into various types:

* Undirected Graph: Edges have no direction (e.g., social networks).
* Directed Graph (Digraph): Edges have a specific direction (e.g., web pages with hyperlinks).
* Weighted Graph: Edges have associated weights (e.g., distances between cities).
* Cyclic Graph: Contains cycles (loops).
* Acyclic Graph: No cycles (e.g., family trees).

**3. Basic Operations on Graphs:**

- **Insertion**: Add nodes or edges to the graph.

- **Deletion**: Remove nodes or edges from the graph.

- **Searching**: Find an entity in the graph.

- **Traversal**: Visit all nodes in the graph.

**4. Applications:**

- **Social Networks**: Representing connections between friends on social media.

- **Computer Networks**: Describing connections between routers and switches.

- **Transportation Networks**: Modeling roads, airports, and routes.

- **Sports Data Science**: Analyzing team dynamics and player interactions.

Graph algorithms help manipulate and analyze graphs, solving problems like finding the shortest path, detecting cycles, and more. They play a crucial role in various fields, including recommendation systems, network analysis, and neural networks.

**5. Advantages:**

Graph data structures provide a powerful way to understand and represent complex relationships in a wide range of applications. Whether it's analyzing team performance or understanding social connections, graphs are a versatile tool for data representation and analysis.