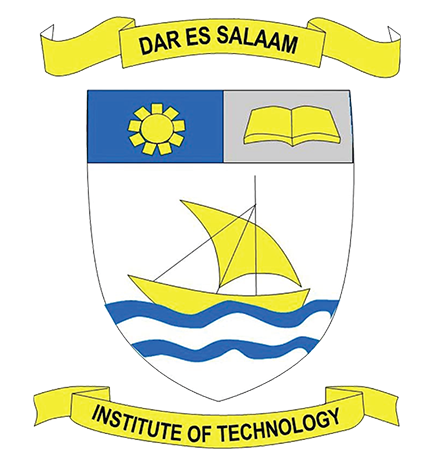
**DAR ES SALAAM INSTITUTE OF TECHNOLOGY (DIT)**

****

**DEPARTMENT OF COMPUTER STUDIES**

**ORDINARY DIPLOMA IN COMPUTER ENGINEERING**

**DATA STRUCTURE FOR TECHNICIANS**

**GROUP ASSIGNMENT**

|  |  |  |
| --- | --- | --- |
| **S/N** | **GROUP MEMBERS** | **REGISTRATION NO** |
| **1** | **NOVATUS KILEO GODWIN** | **220222432813** |
| **2** | **LUCIA MEDARD MWEBESA** | **220222304754** |
| **3** | **KHALID JUMA SALUM** | **220222425015** |

**Introduction**

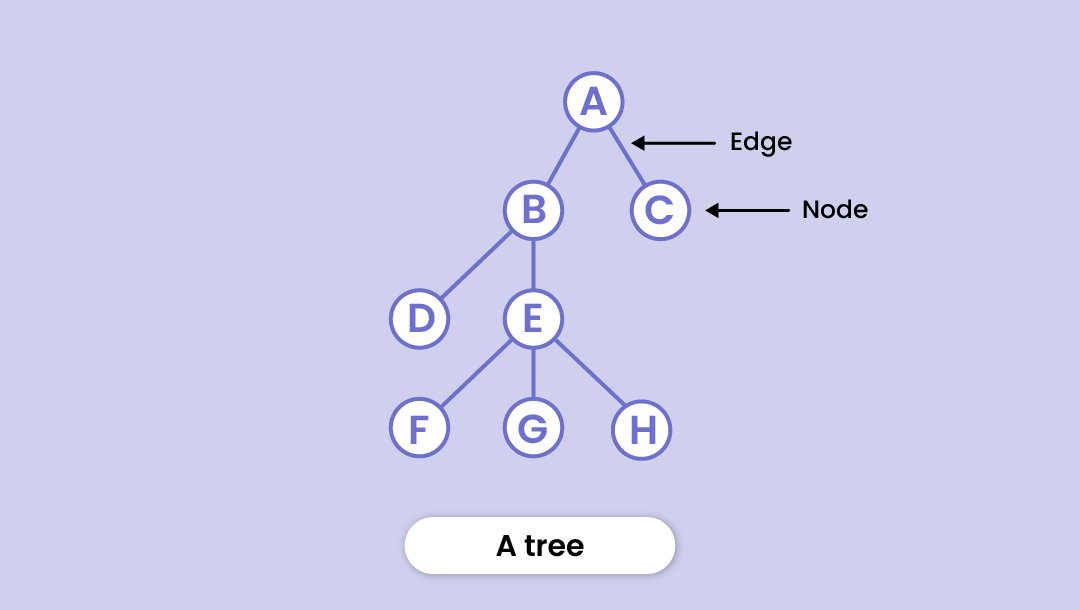
In the realm of computer science, data structures play a vital role in organizing and managing data efficiently. They are used to represent relationships between objects. While they share some similarities, they also have distinct differences that make them suitable for different applications.

**Tree Data Structure**

A tree is a widely used abstract data type that represents a hierarchical tree structure with a set of connected nodes

. Each node in the tree can be connected to many children (depending on the type of tree), but must be connected to exactly one parent, except for the root node, which has no parent

. These constraints mean there are no cycles or "loops" (no node can be its own ancestor), and also that each child can be treated like the root node of its own subtree, making recursion a useful technique for tree traversal



**Key Characteristics of Trees**

Root Node: The topmost node in the tree, which has no parent

Parent Node: The node that is the predecessor of any node

Child Node: A node that is connected below the current one

Leaf Node: A node that has no children

Depth/Level: The length of the path (edges) from the root to a node

Tree Height: The maximum depth from of any node in the tree

**Types of Trees**

**Binary Trees:** A tree where each node has at most two children

**Binary Search Trees (BSTs):** A binary tree with no duplicate nodes that imposes an ordering on its nodes

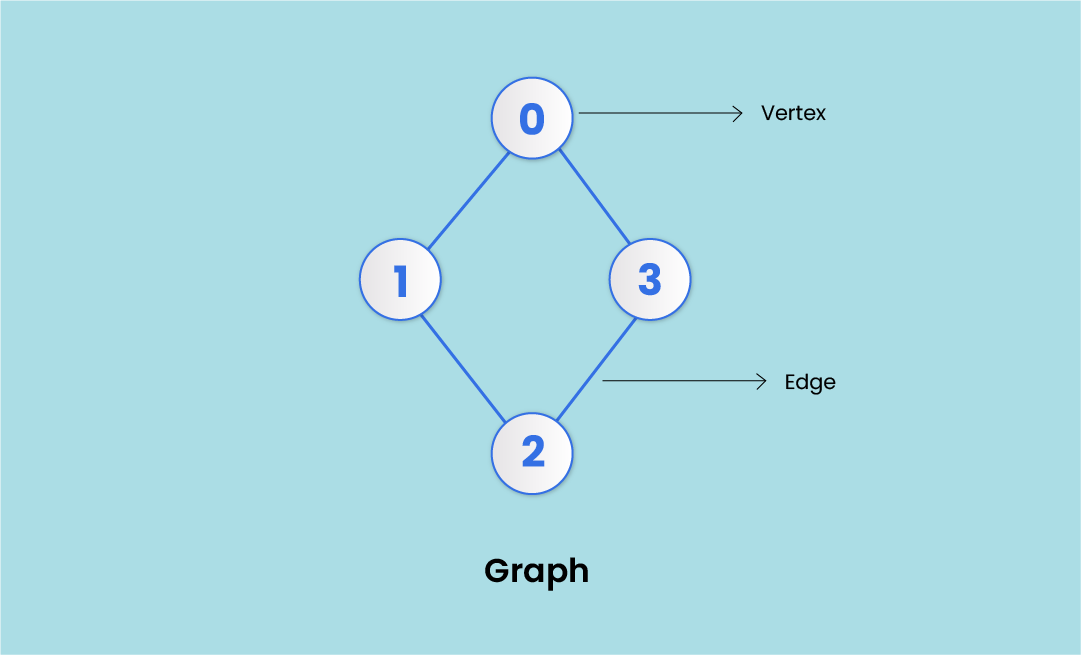
**Graph Data Structure**

A graph is a data structure that contains a set of vertices and a set of edges which connect pairs of the vertices

Graphs are used to depict relationships and links between diverse parts, making it possible to simulate and evaluate complicated systems more efficiently.

**A graph consists of;**

* A collection of nodes also known as vertices, in this case
* A collection of edges E connecting the vertices, (represented as ordered pair of vertices- 0, 1)

-----===

V (Vertices) = {0, 1, 2, 3}

E (Edges) = {(0,1), (0,2), (0,3), (1,2)}

G (Graph) = {V, E}

**Key Characteristics of Graphs**

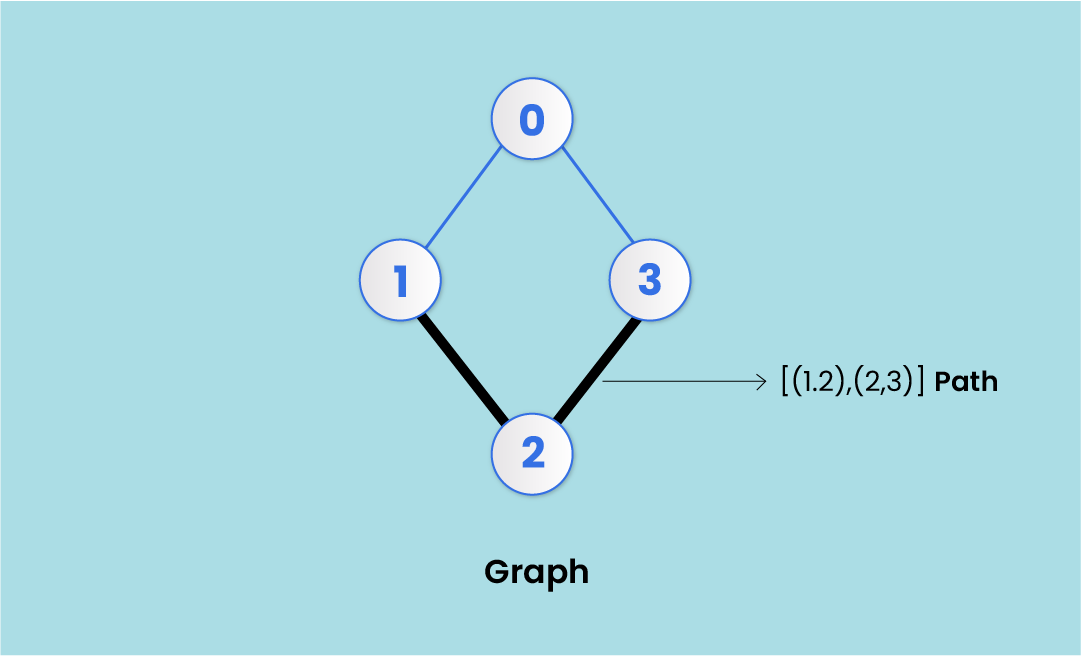
**Vertex**- Each node of a graph is called a vertex. (plural - vertices). In the image above, the dots represented by 0, 1, 2, and 3 are vertices.

**Edge** - Edges in a graph are the lines that connect two vertices. In the image above- the line (0, 1) from 0 to 1 represents an edge, similarly (1, 2), and so on.

Edges can be represented in the form of two-dimensional arrays.

**Adjacency -** A vertex is adjacent to another vertex if there is an edge connecting them. In the image above, vertices 0 and 1 are adjacent, but vertices 1 and 3 are not adjacent as there’s no edge between them.

**Paths -**A series of edges that takes you from one vertex to another when they’re not adjacent to each other. In the above example: 1 and 3 are not adjacent. So, the sequence of edges [(1, 2), (2, 3)] will represent a path.



**Directed Graph** - A graph in which an edge (0, 1) doesn't necessarily mean that there is an edge (1, 0) as well. An edge in a directed graph travels in only one direction. Arrows are used to represent that direction. (More on directed graphs later in the article)

**Basic Operations in a Graph**

**Add/Remove Vertex**

This is the most basic operation in a graph. You have to simply add a vertex to the graph; not needing to connect it with some other vertex through an edge.

While removing a vertex, you have to delete all the edges to and from that particular vertex.

**Add/Remove Edge**

This operation adds or removes an edge between two vertices. If all the edges originating from or ending at say vertex 1 is removed, 1 would become isolated.

**Types of Graphs**

* **Finite graph**: A finite graph has a defined (countable) number of vertices and edges.
* **Infinite graph**: Infinite graph has an unlimited number of vertices and edges.
* **Simple graph**: A simple Graph is one in which there is only one edge connecting a pair of vertices.
* **Null graph**: In a null graph, no edges are there to connect the vertices, meaning, all the vertices are isolated.

**Differences Between Trees and Graphs**

1. **Cycles**: Graphs can contain cycles, whereas trees cannot. This means that in a graph, it is possible for a node to be connected to itself directly or indirectly through other nodes, whereas in a tree, each node can only be connected to other nodes in a hierarchical manner.
2. **Connectivity**: Graphs can be disconnected, meaning they can have multiple components that are not connected to each other. Trees, on the other hand, are always connected, meaning that every node is reachable from every other node.
3. **Hierarchy**: Trees have a clear hierarchical structure, with each node having a single parent node and zero or more child nodes. Graphs do not have this hierarchical structure, and nodes can be connected to each other in a variety of ways.
4. **Root Node**: Trees have a designated root node that has no parent node. Graphs do not have a root node, and nodes can be connected to each other without a specific starting point.
5. **Node Relationships**: In a tree, each node has a specific parent-child relationship. In a graph, the relationships between nodes are arbitrary and can be directed or undirected.
6. **Edge Count**: In a tree, each node can have at most n-1 edges, where n is the number of nodes. In a graph, each node can have any number of edges.
7. **Traversal Complexity**: Traversing a tree is generally simpler and more efficient due to its hierarchical structure. Graphs can have more complex traversal patterns due to the possibility of cycles and disconnected components.
8. **Applications**: Trees are commonly used in hierarchical data structures such as file systems, organizational charts, and XML documents. Graphs are used in a wide variety of applications, including social networks, transportation networks, and computer networks.

**Conclusion**

In conclusion, trees and graphs are two fundamental data structures that have far-reaching implications in various fields of computer science. The hierarchical nature of trees makes them ideal for representing hierarchical relationships, while the non-linear structure of graphs allows for modeling complex systems with interconnected components. As the demand for efficient data processing and analysis continues to grow, the importance of trees and graphs as data structures will only continue to increase. This description has provided a comprehensive overview of the key concepts and applications of trees and graphs, serving as a foundation for further exploration and innovation in the field of computer science.