**Stack**

* **Definition**: A linear data structure that follows the Last In, First Out (LIFO) principle.
* **Operations**:
  + **Push**: Add an element to the top of the stack.
  + **Pop**: Remove and return the top element of the stack.
  + **Peek/Top**: Return the top element without removing it.
  + **isEmpty**: Check if the stack is empty.
* **Use Cases**: Function call management in recursion, expression evaluation and syntax parsing, undo mechanisms in text editors.

**Queue**

* **Definition**: A linear data structure that follows the First In, First Out (FIFO) principle.
* **Operations**:
  + **Enqueue**: Add an element to the end of the queue.
  + **Dequeue**: Remove and return the front element of the queue.
  + **Front/Peek**: Return the front element without removing it.
  + **isEmpty**: Check if the queue is empty.
* **Use Cases**: Order processing, breadth-first search in graphs, task scheduling.

**Singly Linked List**

* **Definition**: A linear collection of nodes where each node contains data and a reference (link) to the next node.
* **Structure**:
  + **Node**: Contains data and a reference to the next node.
  + **Head**: The first node in the list.
  + **Tail** (optional): The last node, whose next reference is **null**.
* **Operations**:
  + **Insertion**: At the beginning, end, or specified position.
  + **Deletion**: Remove a node by value or position.
  + **Traversal**: Visiting each node from the head to the end.
* **Use Cases**: Dynamic memory allocation, implementing other data structures like stacks and queues.

**Doubly Linked List**

* **Definition**: A linked list where each node contains two references: one to the next node and one to the previous node.
* **Structure**:
  + **Node**: Contains data, a reference to the next node, and a reference to the previous node.
  + **Head**: The first node in the list.
  + **Tail**: The last node, which points back to the node before it.
* **Operations**:
  + **Insertion**: At the beginning, end, or specified position.
  + **Deletion**: Remove a node by value or position.
  + **Traversal**: Can be done forward and backward.
* **Use Cases**: More complex data structures like deques, implementing navigation systems (e.g., browsers’ forward and back functionality).

**Circular Linked List**

* **Definition**: A linked list where the last node points back to the first node, forming a circle.
* **Structure**:
  + **Node**: Contains data and a reference to the next node.
  + **Head**: The first node in the list.
  + **Tail**: The last node, whose next reference points to the head.
* **Operations**:
  + **Insertion**: At the beginning, end, or specified position.
  + **Deletion**: Remove a node by value or position.
  + **Traversal**: Can start from any node and will eventually circle back to the starting node.
* **Use Cases**: Implementing round-robin scheduling, circular buffers, and data structures where looping back to the start is necessary.

These data structures are fundamental concepts in computer science, providing the basis for various algorithms and more complex structures.

add trees and graph short notes to above info

ChatGPT

**Stack**

* **Definition**: A linear data structure that follows the Last In, First Out (LIFO) principle.
* **Operations**:
  + **Push**: Add an element to the top of the stack.
  + **Pop**: Remove and return the top element of the stack.
  + **Peek/Top**: Return the top element without removing it.
  + **isEmpty**: Check if the stack is empty.
* **Use Cases**: Function call management in recursion, expression evaluation and syntax parsing, undo mechanisms in text editors.

**Queue**

* **Definition**: A linear data structure that follows the First In, First Out (FIFO) principle.
* **Operations**:
  + **Enqueue**: Add an element to the end of the queue.
  + **Dequeue**: Remove and return the front element of the queue.
  + **Front/Peek**: Return the front element without removing it.
  + **isEmpty**: Check if the queue is empty.
* **Use Cases**: Order processing, breadth-first search in graphs, task scheduling.

**Singly Linked List**

* **Definition**: A linear collection of nodes where each node contains data and a reference (link) to the next node.
* **Structure**:
  + **Node**: Contains data and a reference to the next node.
  + **Head**: The first node in the list.
  + **Tail** (optional): The last node, whose next reference is **null**.
* **Operations**:
  + **Insertion**: At the beginning, end, or specified position.
  + **Deletion**: Remove a node by value or position.
  + **Traversal**: Visiting each node from the head to the end.
* **Use Cases**: Dynamic memory allocation, implementing other data structures like stacks and queues.

**Doubly Linked List**

* **Definition**: A linked list where each node contains two references: one to the next node and one to the previous node.
* **Structure**:
  + **Node**: Contains data, a reference to the next node, and a reference to the previous node.
  + **Head**: The first node in the list.
  + **Tail**: The last node, which points back to the node before it.
* **Operations**:
  + **Insertion**: At the beginning, end, or specified position.
  + **Deletion**: Remove a node by value or position.
  + **Traversal**: Can be done forward and backward.
* **Use Cases**: More complex data structures like deques, implementing navigation systems (e.g., browsers’ forward and back functionality).

**Circular Linked List**

* **Definition**: A linked list where the last node points back to the first node, forming a circle.
* **Structure**:
  + **Node**: Contains data and a reference to the next node.
  + **Head**: The first node in the list.
  + **Tail**: The last node, whose next reference points to the head.
* **Operations**:
  + **Insertion**: At the beginning, end, or specified position.
  + **Deletion**: Remove a node by value or position.
  + **Traversal**: Can start from any node and will eventually circle back to the starting node.
* **Use Cases**: Implementing round-robin scheduling, circular buffers, and data structures where looping back to the start is necessary.

**Tree**

* **Definition**: A hierarchical data structure consisting of nodes, with a single node designated as the root, from which all other nodes branch out.
* **Structure**:
  + **Node**: Contains data and references to child nodes.
  + **Root**: The top node in the tree.
  + **Leaf**: A node with no children.
  + **Parent/Child**: Relationship between nodes directly connected.
  + **Subtree**: A tree consisting of a node and its descendants.
* **Types**:
  + **Binary Tree**: Each node has at most two children.
  + **Binary Search Tree (BST)**: A binary tree with nodes arranged such that the left child has a smaller value and the right child has a larger value.
  + **Balanced Trees (e.g., AVL, Red-Black Trees)**: BSTs that self-balance to ensure O(log n) height.
  + **Heap**: A binary tree where the parent node is greater (max heap) or smaller (min heap) than its children.
  + **Trie**: A tree used for storing a dynamic set of strings.
* **Operations**:
  + **Insertion, Deletion, Traversal (Inorder, Preorder, Postorder)**.
* **Use Cases**: Database indexing (BST), memory management (heaps), autocomplete features (tries).

**Graph**

* **Definition**: A collection of nodes (vertices) and edges connecting pairs of nodes.
* **Types**:
  + **Undirected Graph**: Edges have no direction.
  + **Directed Graph (Digraph)**: Edges have a direction.
  + **Weighted Graph**: Edges have weights/costs associated with them.
  + **Unweighted Graph**: Edges do not have weights.
  + **Cyclic Graph**: Contains at least one cycle.
  + **Acyclic Graph**: No cycles (DAGs - Directed Acyclic Graphs).
* **Representation**:
  + **Adjacency Matrix**: 2D array to represent edges.
  + **Adjacency List**: Array of lists to represent edges.
* **Operations**:
  + **Traversal**: Depth-First Search (DFS), Breadth-First Search (BFS).
  + **Shortest Path Algorithms**: Dijkstra’s, Bellman-Ford.
  + **Minimum Spanning Tree**: Kruskal’s, Prim’s.
* **Use Cases**: Network routing, social networks, dependency resolution, geographical mapping (GPS).

These data structures are foundational in computer science, enabling efficient data storage, manipulation, and retrieval for a wide range of applications.

make the info in the form of bullets

ChatGPT

**Stack**

* **Definition**: Linear data structure; Last In, First Out (LIFO).
* **Operations**:
  + **Push**: Add element to top.
  + **Pop**: Remove and return top element.
  + **Peek/Top**: Return top element without removing.
  + **isEmpty**: Check if stack is empty.
* **Use Cases**: Recursion, expression evaluation, undo mechanisms.

**Queue**

* **Definition**: Linear data structure; First In, First Out (FIFO).
* **Operations**:
  + **Enqueue**: Add element to end.
  + **Dequeue**: Remove and return front element.
  + **Front/Peek**: Return front element without removing.
  + **isEmpty**: Check if queue is empty.
* **Use Cases**: Order processing, breadth-first search, task scheduling.

**Singly Linked List**

* **Definition**: Linear collection of nodes; each node links to next.
* **Structure**:
  + **Node**: Contains data and next reference.
  + **Head**: First node.
  + **Tail**: Last node (optional, points to **null**).
* **Operations**:
  + **Insertion**: At beginning, end, or specified position.
  + **Deletion**: By value or position.
  + **Traversal**: Visit nodes from head to end.
* **Use Cases**: Dynamic memory allocation, stacks, queues.

**Doubly Linked List**

* **Definition**: Nodes with references to next and previous nodes.
* **Structure**:
  + **Node**: Contains data, next and previous references.
  + **Head**: First node.
  + **Tail**: Last node.
* **Operations**:
  + **Insertion**: At beginning, end, or specified position.
  + **Deletion**: By value or position.
  + **Traversal**: Forward and backward.
* **Use Cases**: Deques, browser navigation.

**Circular Linked List**

* **Definition**: Last node points to first, forming a circle.
* **Structure**:
  + **Node**: Contains data and next reference.
  + **Head**: First node.
  + **Tail**: Last node points to head.
* **Operations**:
  + **Insertion**: At beginning, end, or specified position.
  + **Deletion**: By value or position.
  + **Traversal**: Loop from any node back to start.
* **Use Cases**: Round-robin scheduling, circular buffers.

**Tree**

* **Definition**: Hierarchical structure; nodes with parent-child relations.
* **Structure**:
  + **Node**: Contains data and child references.
  + **Root**: Top node.
  + **Leaf**: Node with no children.
  + **Parent/Child**: Directly connected nodes.
  + **Subtree**: Node with its descendants.
* **Types**:
  + **Binary Tree**: Max two children per node.
  + **BST**: Left child < parent, right child > parent.
  + **Balanced Trees**: Self-balancing (e.g., AVL, Red-Black).
  + **Heap**: Max heap (parent > children), min heap (parent < children).
  + **Trie**: Stores dynamic set of strings.
* **Operations**:
  + **Insertion, Deletion, Traversal (Inorder, Preorder, Postorder)**.
* **Use Cases**: Database indexing, memory management, autocomplete.

**Graph**

* **Definition**: Collection of nodes (vertices) and edges.
* **Types**:
  + **Undirected**: Edges have no direction.
  + **Directed (Digraph)**: Edges have direction.
  + **Weighted**: Edges have weights/costs.
  + **Unweighted**: Edges have no weights.
  + **Cyclic**: Contains cycles.
  + **Acyclic (DAGs)**: No cycles.
* **Representation**:
  + **Adjacency Matrix**: 2D array.
  + **Adjacency List**: Array of lists.
* **Operations**:
  + **Traversal**: DFS, BFS.
  + **Shortest Path**: Dijkstra’s, Bellman-Ford.
  + **Minimum Spanning Tree**: Kruskal’s, Prim’s.
* **Use Cases**: Network routing, social networks, dependency resolution, GPS mapping.