Data Structure Visualizer - Project Documentation

1. Project Purpose

This project is an **Educational Tool** designed for Computer Science students. Its primary goal is to provide a visual representation of common data structures and their operations. By visualizing how data is inserted, deleted, and organized in memory, students can gain a deeper understanding of the underlying logic and complexity of these structures, which are fundamental to the field of Computer Science.

2. Module Breakdown

The project is organized into `Core` logic (data structure implementations) and `UI` (visualization logic).

A. Linear Data Structures

These structures organize data sequentially.

1. Stack (LIFO - Last In First Out)

- **Implementation**: Uses Java's built-in `java.util.Stack`.
- **Primary Logic File**: src/ui/panels/StackVisualizerFrame.java
- **Functionality**: Visualizes pushing and popping elements from a stack.

2. Queue (FIFO - First In First Out)

- **Implementation**: Uses Java's built-in `java.util.LinkedList` (as a Queue).
- **Primary Logic File**: src/ui/panels/QueueVisualizerFrame.java
- **Functionality**: Visualizes enqueuing at the rear and dequeuing from the front.

3. Singly Linked List

- **Implementation**: Uses Java's built-in `java.util.LinkedList`.
- **Primary Logic File**: src/ui/panels/LinkedListVisualizerFrame.java
- **Functionality**: Visualizes standard list operations. Note that while it visualizes a Singly Linked List, the underlying Java implementation is doubly linked. The visualization restricts operations to mimic singly linked behavior where appropriate.

4. Array / ArrayList

- **Implementation**: Uses Java's built-in `java.util.ArrayList`.

- **Primary Logic File**: src/ui/panels/ArrayVisualizerFrame.java
- **Functionality**: Visualizes indexed access, insertion, and deletion in a dynamic array.

5. Doubly Linked List

- **Implementation**: **Custom Implementation**.
- **Primary Logic File**: src/core/structures/DoublyLinkedList.java
- **Functionality**: A list where nodes point to both next and previous nodes, allowing bidirectional traversal.

6. Circular Linked List

- **Implementation**: **Custom Implementation**.
- **Primary Logic File**: src/core/structures/CircularLinkedList.java
- **Functionality**: A list where the last node points back to the head, forming a circle.

B. Non-Linear Data Structures

These structures organize data hierarchically or interconnectedly.

1. Binary Search Tree (BST)

- **Implementation**: **Custom Implementation**.
- **Primary Logic File**: src/core/structures/BinarySearchTree.java
- **Functionality**: A tree where the left child is smaller and the right child is larger than the parent. Includes traversals.

2. Graph

- **Implementation**: **Custom Implementation** (Adjacency List).
- **Primary Logic File**: src/core/structures/SimpleGraph.java
- **Functionality**: Represents connections between nodes (vertices) using edges. Includes traversal algorithms.

---

3. Key Methods & Functionality

Linear Data Structures

src/core/structures/DoublyLinkedList.java

Method	Functionality
insertFirst(int value)	Creates a new node and adds it to the start of the list. Updates `head` and `prev` pointers.
insertLast(int value)	adds a new node to the end. Updates `tail` and `next` pointers.
insertAt(int index, int value)	Inserts a node at a specific index by traversing to the position.
deleteFirst()	Removes the head node and updates the new head's `prev` to null.
deleteLast()	Removes the tail node and updates the new tail's `next` to null.
search(int value)	Traverses the list to find the index of a value.

src/core/structures/CircularLinkedList.java

Method	Functionality
insertFirst(int value)	Inserts at the head and updates the last node's `next` pointer to point to the new head.
insertLast(int value)	Inserts at the end and updates the new node's `next` to point to `head`.
deleteFirst()	Removes head and updates the last node to point to the new head.
deleteLast()	Traverses to the second-to-last node and updates its `next` to `head`.

src/ui/panels/StackVisualizerFrame.java (Logic Wrapper)

Method	Functionality
push()	Pushes a value onto the `java.util.Stack`. Checks for overflow (visual limit).
pop()	Pops the top element. Checks for underflow.
peek()	Inspects the top element without removing it.

src/ui/panels/QueueVisualizerFrame.java (Logic Wrapper)

Method	Functionality
enqueue()	Adds an element to the Queue (using `offer`). Checks for capacity.
dequeue()	Removes an element from the front (using `poll`).

Non-Linear Data Structures

src/core/structures/BinarySearchTree.java

Method	Functionality
insert(int value) / insertRec	Recursively finds the correct position and inserts a new node to maintain BST property.
delete(int value) / deleteRec	Recursively finds and removes a node. Handles 3 cases: leaf node, one child, or two children (finding inorder successor definition).
search(int value) / searchRec	Recursively checks left or right subtrees to find if a value exists.

src/core/structures/SimpleGraph.java

Method	Functionality
addNode(int node)	Adds a new vertex to the adjacency map.
addEdge(int from, int to)	Adds a directed edge from `from` to to in the adjacency list.
getNeighbors(int node)	Returns the list of connected nodes for a given vertex.

Algorithms

src/core/algorithms/TreeAlgorithms.java

Method	Functionality
inorder(TreeNode root)	Returns string of values in Left-Root-Right order (Sorted for BST).
preorder(TreeNode root)	Returns string of values in Root-Left-Right order.

Method	Functionality
postorder(TreeNode root)	Returns string of values in Left-Right-Root order.

src/core/algorithms/GraphAlgorithms.java

Method	Functionality
bfs(SimpleGraph graph, int start)	Breadth-First Search. Uses a Queue and `Set<Visited>` to explore layer by layer.
dfs(SimpleGraph graph, int start)	Depth-First Search. Uses recursion and `Set<Visited>` to explore as deep as possible first.