PROJECT 2

Introduction:

Data structures are the backbone of programming, providing a framework for organizing and accessing data efficiently. Regardless of the programming paradigm employed, mastery of data structures is indispensable for developing effective algorithms and software solutions. This project endeavors to identify and describe key data structures commonly used in programming, followed by their implementation in Java. By delving into the intricacies of these foundational structures, we aim to deepen our understanding of programming principles and enhance our ability to craft elegant and efficient code solutions.

(A) Data Structures:

1. Arrays:

- Arrays are collections of elements stored in contiguous memory locations, allowing for random access to elements using indices.
- Arrays provide constant-time access to elements but have linear-time complexity for insertion and deletion operations.

2. Linked Lists:

- Linked lists are linear data structures consisting of nodes, where each node contains data and a reference (or link) to the next node in the sequence.
- Linked lists support constant-time insertion and deletion at both ends but require linear-time traversal for access.

3. Stacks:

- Stacks are linear data structures that follow the Last In, First Out (LIFO) principle, where elements are inserted and removed from the same end, known as the top.
- Stacks support constant-time insertion and deletion operations.

4. Queues:

- Queues are linear data structures that follow the First In, First Out (FIFO) principle, where elements are inserted at the rear and removed from the front.
- Queues support constant-time insertion and deletion operations.

5. Binary Trees:

- Binary trees are hierarchical data structures consisting of nodes, where each node has at most two children: left and right.
- Binary trees support efficient search, insertion, and deletion operations, typically with logarithmic time complexity.

6. Graphs:

- Graphs are non-linear data structures consisting of vertices (nodes) and edges (connections between vertices).
- Graphs can be represented using adjacency lists, adjacency matrices, or other methods, and support various operations such as traversal and path finding.

7. Hash Tables:

- Hash tables are data structures that store key-value pairs and use a hash function to compute an index for efficient data retrieval.
- Hash tables provide constant-time average-case complexity for insertion, deletion, and retrieval operations.

(B) Development in Java:

Below are basic implementations of the identified data structures in Java:

```
// Array implementation
public class MyArray {
    private int[] array;
    private int size;
```

```
    public MyArray(int capacity) {
        array = new int[capacity];
        size = 0;
    }

    // Implement methods for insertion, deletion, access, etc.
}

// Linked list implementation
public class ListNode {
    int val;
    ListNode next;

    public ListNode(int val) {
        this.val = val;
    }
}

public class MyLinkedList {
    private ListNode head;

    // Implement methods for insertion, deletion, traversal, etc.
}

// Stack implementation
public class MyStack {
    private int[] array;
    private int top;

    // Implement methods for push, pop, peek, etc.
}

// Queue implementation
public class MyQueue {
```

```
    private int[] array;
    private int front;
    private int rear;

    // Implement methods for enqueue, dequeue, peek, etc.
}

// Binary tree implementation
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
    }
}

public class BinaryTree {
    private TreeNode root;

    // Implement methods for insertion, deletion, traversal, etc.
}

// Graph implementation (using adjacency list)
import java.util.*;

public class Graph {
    private int V;
    private LinkedList<Integer>[] adjList;

    public Graph(int V) {
        this.V = V;
        adjList = new LinkedList[V];
        for (int i = 0; i < V; i++) {
```

```
            adjList[i] = new LinkedList<>();
        }
    }

    // Implement methods for adding edges, traversal, etc.
}

// Hash table implementation
public class MyHashTable {
    private int capacity;
    private int[] table;

    public MyHashTable(int capacity) {
        this.capacity = capacity;
        table = new int[capacity];
    }

    // Implement methods for insertion, deletion, searching, etc.
}
```

Conclusion:

In conclusion, a thorough understanding of data structures is essential for efficient programming. The provided implementations in Java serve as foundational building blocks for developing robust and optimized software applications.