Learning Objectives

Learners will be able to…

- **Define a data structure**
- **Cover Basic operations of data structures**
- **Perform Basic Operations with arrays**
- **Differentiate Linear and Non-Linear Data Structures**

Data Structures

What is a Data Structure?

Let's break down the two fundamental words in **Data Structures**: `Data` and `Structure`.

Data: In the simplest terms, data is information. In the context of computer science, `data` refers to the quantities, characters, or symbols on which operations are performed by a computer. `Data` can exist in various forms - it could be numeric values, text, images, audio, video, and even complex structured types like lists, arrays, and so forth.

Structure: Structure implies the arrangement or organization of components. In the world of computer science, when we refer to the `structure`, we're talking about how we are organizing or arranging our data in the computer's memory for efficient access and modification.

definition

Data Structures are particular ways of organizing data in a computer so that we can perform different operations. Some of those basic operations include:

- Insertion
- Deletion
- Merging
- Searching
- Sorting
- Traversal

We can categorize data types into two categories:

- **Primitive data types**
- **Non-primitive data types**

The primitive type of data types includes the predefined data structures such as char, float, int, and double. Non-Primitive data types are composed of primitive data types.

Simple data structures are types of data structures that are generally built from primitive data types like `int`, `float`, `double`, `string`, `char`.

Compound data structures are types that the user can build by combining simple data structures.

The non-primitive data structures are used to store the collection of elements. This data structure can be further categorized into **Linear and Non-Linear data structures**.

Linear data structure – It is a type of data structure where the arrangement of the data follows a linear fashion. It is a type of data structure where data is stored and managed in a linear sequence. Some of the popular linear data structures that we widely use in Java are arrays, stacks, queues, and linked lists

Non-Linear data Structures – It is a type of data structures are basically multilevel data structures. the data are arranged in a manner that does not follow a linear fashion. Some of the popular non-linear data structures that we widely use are trees and graphs.

Algorithms

Algorithms

Now that we are more or less familiar with Data structures bit of “data structures and algorithms”. We can tackle the question “what is an algorithm?”

definition

An **algorithm** is a step-by-step procedure or a set of rules to be followed in calculations or other problem-solving operations, especially by a computer. In simpler terms, it’s a recipe for accomplishing a task.

For example, an algorithm could be a process for sorting a list of names alphabetically, calculating the square root of a number, finding the shortest path in a graph, or any other sequence of operations that can be programed and implemented.

The study of algorithms and data structures go hand in hand. Data structures are often designed to facilitate specific algorithms, and algorithms are often created to process specific data structures. Together, they form the foundation of computer programming and are essential for the design of efficient software.

Lets try an example with arrays that we should be familiar with. An array is a data structure, therefore we are looking to be able to perform the following operation to it: traversal, insertion, deletion, merging.

We are going to try a basic array traversal. Our step would be given an array we will check each element one by one until the very last element. Our recipe in this case will be to use a loop and traverse our array.

```java
public class ArrayTraversal {

    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};

        // Iterate through the array using a for loop.
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
        }
    }
}
```

In this course, “**Data Structures and Algorithms”**, we will explore the symbiotic relationship between these two essential aspects of computer science. We’ll dive deeper into various types of data structures, understand their internal workings, learn when to use which data structure, and examine their space and time complexities. For the rest of this lesson we will tackle different operations with the array data structure.

Insertion

Insertion in Arrays

When it comes to arrays, **insertion** refers to the process of adding a new element to the array at a specific position. However, since arrays in Java are of fixed size, we don't have a direct method to add an element to it.

To perform an insertion operation, we usually have to define a new array with a size larger than the original array by one and then copy elements from the old array to the new one. After copying, we can add the new element at the desired location.

Here's an example of how you could do it:

```java
public class ArrayInsertion {

    public static void main(String[] args) {

        int[] array = {1, 2, 3, 4, 5};
        int insertValue = 10;
        int insertIndex = 2;

        // print original array
        System.out.println("Original Array:");
        for (int i : array) {
            System.out.print(i + " ");
        }
        System.out.println();

        // create a new array of size array.length + 1
        int[] newArray = new int[array.length + 1];

        // copy elements from old array to new array, leaving
        space for new element
        for (int i = 0; i < insertIndex; i++)
            newArray[i] = array[i];

        // add new element
        newArray[insertIndex] = insertValue;

        // copy the rest of elements from old array to new array
        for (int i = insertIndex + 1; i < newArray.length; i++)
            newArray[i] = array[i - 1];

        // print new array
        System.out.println("Array after insertion:");
        for (int i : newArray) {
            System.out.print(i + " ");
        }
    }
}
```

However, this method is not efficient, especially for large arrays or for multiple insertions. We would typically use a dynamic data structure, such as an ArrayList in Java, for these types of operations, which we will explore later in the course.

Deletion

Deletion in Arrays

Deletion in an array refers to the operation of removing an element from the array. Like insertion, deletion is also tricky with arrays because arrays in Java are of a fixed size.

To delete an element from an array, we don’t have a direct method. Instead, we’ll typically create a new array with a size smaller than the original array by one, and then copy all elements excluding the one to be deleted from the old array to the new one.

Let’s look at an example where we delete an element from a specific index in an array:

```java
public class ArrayDeletion {

    public static void main(String[] args) {

        int[] array = {1, 2, 3, 4, 5};
        int deleteIndex = 2;

        // print original array
        System.out.println("Original Array:");
        for (int i : array) {
            System.out.print(i + " ");
        }
        System.out.println();

        // create a new array of size array.length - 1
        int[] newArray = new int[array.length - 1];

        // copy elements from old array to new array excluding
        the element to be deleted
        for (int i = 0, k = 0; i < array.length; i++) {

            // this condition will skip the element to be
        deleted
            if (i == deleteIndex) {
                continue;
            }

            // if the index is not the deletion index, add the
        element to the new array
            newArray[k++] = array[i];
        }

        // print new array
        System.out.println("Array after deletion:");
        for (int i : newArray) {
            System.out.print(i + " ");
        }
    }
}
```

As with insertion, this method is not very efficient, especially for large arrays or multiple deletions. It involves shifting the elements and reallocating the memory which takes a lot of time and resources. This issue is one of the reasons why we often use more complex data structures like linked lists or dynamic arrays (ArrayList in Java) for tasks involving frequent insertions and deletions. We will learn about these data structures later in this course.

Merging

Merging Two Arrays

Merging in the context of arrays involves combining two arrays into a single array. Unlike insertion and deletion, merging does not alter the original arrays but creates a new one that holds all elements from both arrays.

The simplest way to merge two arrays in Java is to first create a new array of the correct size (which should be the sum of the lengths of the two input arrays), then copy over the elements from both arrays.

Here is an example:

```java
public class ArrayMerge {

    public static void main(String[] args) {

        int[] array1 = {1, 2, 3, 4, 5};
        int[] array2 = {6, 7, 8, 9, 10};

        // create a new array of size array1.length +
        array2.length
        int[] mergedArray = new int[array1.length +
        array2.length];

        // copy elements from array1 to the new array
        for (int i = 0; i < array1.length; i++)
            mergedArray[i] = array1[i];

        // copy elements from array2 to the new array
        for (int i = 0; i < array2.length; i++)
            mergedArray[i + array1.length] = array2[i];

        // print merged array
        System.out.println("Merged Array:");
        for (int i : mergedArray) {
            System.out.print(i + " ");
        }
    }
}
```

In this code, we first create a new array of size equal to the sum of sizes of the input arrays. Then we copy all elements from the first array followed by all elements from the second array into the new array. As a result, the new array contains all elements from both input arrays in their original order.

However, this approach does not take into account any sorting or ordering of elements in the input arrays. If your input arrays are sorted and you want the merged array to be sorted as well, you’ll need to implement a more complex merging algorithm, such as the one used in merge sort, or sort the merged array afterwards.

We’ll explore such algorithms in later parts of this course, where we’ll learn about sorting algorithms and their applications.

Formative Assessment 1

Formative Assessment 2