Course Title: Data Structure

Lecture: 1
Date: 06 June 2022

Data Structure

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are widely used in almost every aspect of Computer Science i.e., Operating System, Compiler Design, Artificial intelligence, Graphics and many more.

Data Structures are the main part of many computer science algorithms as they enable the programmers to handle the data in an efficient way. It plays a vital role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible.

Basic Terminology

Data: Data are simply values or set of values. A data item refers to a single unit of values. Social security number would normally be treated as a single data element.

Group Items: Data items that are divided into subitems are called group items. An employee name may be divided into three subitems: first name, middle name, and last name.

Record: Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

File: A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

Attribute and Entity: An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity. For example, the following are possible attribute and their corresponding values for an entity an employee of a given organization:

Attributes:	Name	Age	Sex	Social Security Number
Values:	Jakaria	26	M	1234

Information: The term information is sometimes used for data with given attributes or in other words meaningful and processed data.

Advantages of Data Structures

Efficiency: Efficiency of a program depends upon the choice of data structures. For example: suppose, we have some data, and we need to perform the search for a particular record. In that case, if we organize our data in an array, we will have to search sequentially element by element. hence, using array may not be very efficient here. There are better data structures which can make the search process efficient like ordered array, binary search tree or hash tables.

Reusability: Data structures are reusable, i.e., once we have implemented a particular data structure, we can use it at any other place. Implementation of data structures can be compiled into libraries which can be used by different clients.

Abstraction: Data structure is specified by the ADT which provides a level of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.

What are abstract data types (ADT)?

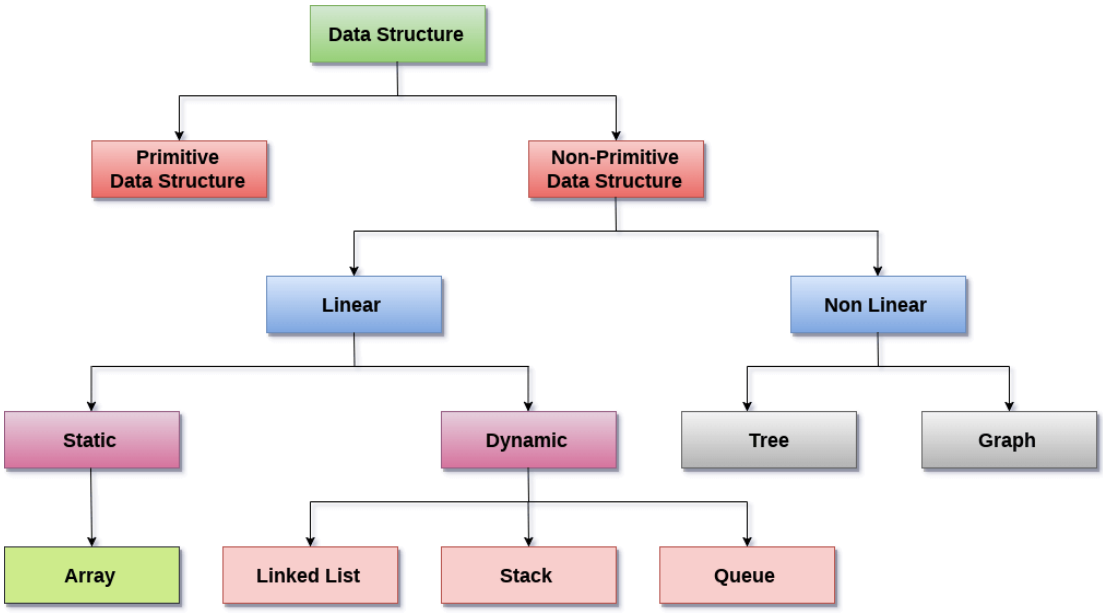
Remember the goal of software development? Robustness, adaptability, and reusability. Out of this effort to write better code arose a new metaphor for using and building data structures: *abstract data type*, which highlights the notion of *abstractness*.

When we say, "data type", we often refer to the primitive data types built into a language, such as integer, real, character, and Boolean. An integer is most likely implemented or represented in four bytes in the computer. However, when we use integers, we do not worry at all about its internal representation, or how these operations are implemented by the compiler in machine code. Additionally, we know that, even when we run our program on a different machine, the behavior of an integer does not change, even though its internal representation may change. What we know is that we can use primitive data types via their operational interface -- '+', '-', '*' and '/' for integers. The primitive data types were abstract entries.

A stack or a queue is an example of an ADT. Both stacks and queues can be implemented using an array. It is also possible to implement stacks and queues using linked lists. This demonstrates the "abstract" nature of stacks and queues: how they can be considered separately from their implementation.

Applying the idea of abstraction to data structures, we have ADT for data structures. On the one hand, an ADT makes a clean separation between interface and implementation, the user only sees the interface and therefore does not need to tamper with the implementation. On the other hand, if the implementation of an ADT changes, the code that uses the ADT does not break, since the interface remains the same. Thus, the abstraction makes the code more robust and easier to maintain. Moreover, once an ADT is built, it may be used multiple times in various contexts. For example, the list ADT may be used directly in application code, or may be used to build another ADT, such as a stack.

Classification of Data Structure:



Linear data structure: Data structure where data elements are arranged sequentially or linearly where each element is attached to its previous and next adjacent is called a linear data structure.
Examples of linear data structure are array, stack, queue, linked list etc.

- **Static data structure:** Static Data structure has fixed memory size. It is easier to access the element in static data structure.
  Examples of this data structure is array.
- **Dynamic data structure:** In Dynamic Data Structure the size is not fixed, the size can be randomly updated during run time which may be considered efficient with respect to memory complexity of the code.
  Examples of this data structure is queue, stack, etc.

Non-linear data structure: Data structure where data elements are not sequentially or linearly are called non-linear data structures. In non-linear data structure we can't traverse all the elements in single run only.
Examples of non-linear data structure are trees and graphs.

Operations on data structure

1) **Traversing:** Every data structure contains the set of data elements. Traversing the data structure means visiting each element of the data structure to perform some specific operation like searching or sorting.

Example: If we need to calculate the average of the marks obtained by a student in 6 different subjects, we need to traverse the complete array of marks and calculate the total sum, then we will divide that sum by the number of subjects i.e., 6, to find the average.

2) **Insertion:** Insertion can be defined as the process of adding the elements to the data structure at any location.

If the size of data structure is **n** then we can only insert **n-1** data elements into it.

3) **Deletion:** The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location.

If we try to delete an element from an empty data structure, then **underflow** occurs.

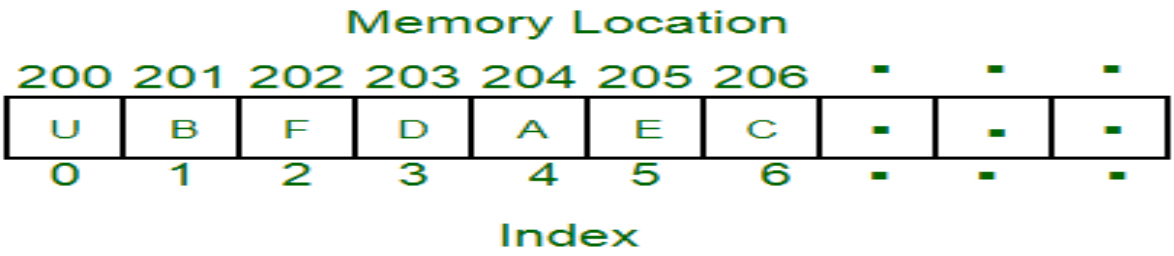
4) **Searching:** The process of finding the location of an element within the data structure is called Searching. There are two algorithms to perform searching, Linear Search and Binary Search. We will discuss each one of them later in this tutorial.

5) **Sorting:** The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

6) **Merging:** When two lists List A and List B of size M and N respectively, of similar type of elements, clubbed or joined to produce the third list, List C of size (M+N), then this process is called merging.

Arrays:

An array is a linear data structure, and it is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. It allows the processing of a large amount of data in a relatively short period. It stores multiple data of the same type at one time. The first element of the array is indexed by a subscript of 0. There are different operations possible in Array like Searching, Sorting, Inserting, Traversing, Reversing, and Deleting.



Characteristics of an Array:

An array has various different characteristics which is as follows:

- Arrays uses an index-based data structure which helps to identify each of the elements in an array easily using the index.
- If user want to store multiple values of same data type, then array can be and utilized efficiently.
- Array can also handle complex data structure by storing data in two-dimensional array.
- Array is also used to Implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
- The search process in array can be done very easily.
- In array accessing an element is very easy by using the index number.

Applications of Array:

Different applications of Array are as follow:

- Array is used in solving matrices problem
- Databases records are also implemented by array.
- It helps in implementing sorting algorithm.
- It is also used to Implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
- Array can be used for CPU scheduling.
- Applied as lookup table in computer.
- Arrays can be used in speech processing, where every speech signal is an array.

Real Life Applications of Array:

- Array is frequently used to store data for mathematic computations.
- It is used in image processing.
- It is also used in record management.
- Book pages is also real-life example of array.
- It is used to ordering boxes also.

Linear Search:

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

Algorithm:

Linear Search (Array A, Value x)

Step 1: Set i to 0
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit

Implementation on Python:

```
arr = [2, 3, 9, 10, 40]
x = 90
n = len(arr)


def search(arr, n, x):

    for i in range(0, n):
        if (arr[i] == x):
            return i
    return -1



# Function call
result = search(arr, n, x)
if(result == -1):
    print("Element is not present in array")
else:
    print("Element is present at index", result)
```

```
Element is not present in array
```

Bubble Sort

Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order. It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water. Bubbles in water rise to the surface; similarly, the array elements in bubble sort move to the end in each iteration. Although it is simple to use, it is primarily used as an educational tool because the performance of bubble sort is poor in the real world. It is not suitable for large data sets. The average and worst-case complexity of Bubble sort is **O(n^2),** where **n** is a number of items.

```
a = [35, 10, 31, 11, 26]
print("Before sorting array elements are - ")
for i in a:
    print(i, end = " ")
for i in range(0,len(a)):
    for j in range(i+1,len(a)):
        if a[j]<a[i]:
            temp = a[j]
            a[j]=a[i]
            a[i]=temp
print("\nAfter sorting array elements are - ")
for i in a:
    print(i, end = " ")
```

```
Before sorting array elements are -
35 10 31 11 26
After sorting array elements are -
10 11 26 31 35
```