gradeup

Data Structure Part-1

Content:

Elementary Data Organization:

Data are nothing but some type of a statistic or a set of values (statistics). A data item refers to a single unit of data. Data can be either raw or processed. Raw data can be divided into two subparts i.e., group items and elementary items. For example name of a person may be assumed as group item as it is a combination of first name, middle name and surname. While employee number in an employee record cannot be divided further that's why known as elementary data item.

If one arrange some data in an appropriate sequence, then it forms a Structure and gives us a meaning. This meaning is known as information. Information is also called as processed data.

Data can be represented in a hierarchy in a computer science. This hierarchy comprises of field, records and files. For understanding this hierarchy let us introduce some more concepts.

An entity is something that has certain properties or attributes which may be assigned some value which can be either numeric or non - numeric. For example, an employee

may be assumed as an entity, while its attributed are name, age, sex, etc. entities with similar attributes such as all the employees in a university, forms an entity set.

The way the data are organized into hierarchy as above reflects the relationship between attributes, entities and entity sets. Thus we can say that, a field is a single elementary unit of information representing an attribute of an entity, a record is collection of field values of a given entity and a file is a collection of records of the entities in a given set.

This organization of data into fields, records and files is not sufficient to maintain and process all types of collection of data efficiently. Due to this reason, data are also organized into more complex types of structures. The study of these complex types of structures from the basis for data structures and comprise of following three steps:

1. Logical or mathematical representation of structure
2. Implementation of the structure on a computer
3. Quantitative analysis of the structure, which determines time and space complexity for the processing of the structure.

Data Structure

Data structure can be defined in a number of ways, some of the definitions of data structures are as follows:

1. A data structure is a systematic way of accessing and organizing data.
2. A data structure tries to structure data:
   a. Usually more than one piece of data
   b. Should define legal operations on the data
   c. The data might be grouped together (e.g. in an linked list)
3. When we defined a data structure we are in fact creating a new data type of our own.
   a. Using predefined types or previously user defined types.
   b. Such new types are then used to reference variables type within a program.

But the most useful definition of data structure is: "the logical or mathematical model of a particular organization of data is called a data structure".

The choice of a particular data model depends basically on two considerations. First, it must be rich enough to represent the actual relationship of data in real world. Second, the structure should simple enough that can effectively process the data whenever required.

Why Data Structure?

Following are some of the reasons for using the data structures:

- It study how data are stored in a compute so that operations can be implemented efficiently
- Data structures are especially important when you have a large amount of information
- Concrete and Conceptual ways to organize data for efficient storage and manipulation.

TYPES OF DATA STRUCTURE

Data structures can be categorized in two different ways viz. Primitive and Non-Primitive (Composite) data structure.

Primitive Data Structure: Basically primitive data structure are not data structures instead they are termed as primitive data types. It's basic data structure and it's directly operated upon the machine instructions. These structures have different representation on different computes. In computer science, primitive data type can refer to either of the following mentioned concepts:

- A basic type is a data type provided by a programming language as a basic building block of that language. Most languages allow more complicated composite type to be recursively constructed starting from basic types.
- A built-in-type is a data type for which the programming language provide built-in support.

Classic basic primitive types may include:

1. Integers
2. Floating-point number
3. Characters
4. Pointers
5. Boolean

Composite (Non – Primitive) Data Structure: Composite data structure is based on the primitive data structure. While primitive data structures are the basic building blocks, composite data structures are created using these primitive data structures.

Composite data structure can again be divided into two categories:

1. Linear Data Structure
2. Non – Linear Data Structure

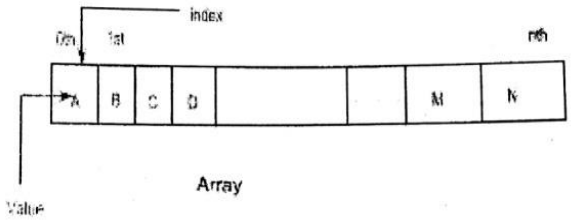
Let's discus them one by one.

Linear Data Structure: Linear data structure is said to be its elements or items from a sequence one after other. Linear data structure comprises various data structures such as string, array, linked list, stack, queue, etc.

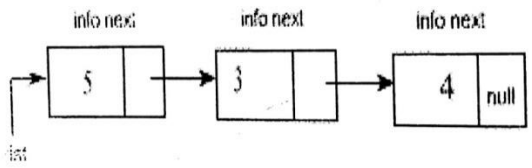
✓ **String:** It is, essentially, a sequence of characters. A string is generally understood as a data type storing a sequence of data values, usually bytes, in which elements usually stand for characters according to a character encoding, which differentiates it from the more general array data type.

✓ **Array:** In computer programming, a group of homogeneous elements of a specific data type is known as an array, one of the simplest data structures. It hold a series of data elements, usually of the both i.e. same size and data type. Individual elements are accessed by their position in a array. The position is given by a index, which is also called a subscript. The index usually uses a consecutive range of integers, but the index can have any ordinary set of values.

Some arrays are multi-dimensional, meaning they are indexed by a fixed number of integers. Generally, one - and two - dimensional arrays are the most common. Most programming languages have a built - in - array data types.



✓ **Link List:** In computer science, a linked list is one of the fundamental data structures used in computer programming. It consists of a sequence of nodes, each containing arbitrary data fields and one or two references pointing to the next and/or previous nodes. A linked list is a self-referential data type because it contains a linked to another data of the same type. Linked lists permit insertion and removal of nodes at any point in the list in linear time, but do not allow random access.



There are various types of linked list such as Singly - linked List, Doubly - Linked List, Circular - Linked List, Header Linked List, etc.

✓ **Stack:** A stack is a linear Structure in which item may be added or removed only at one end. There are certain frequent situations in computer science when one wants to restrict insertions and deletions so that they can take place only at the beginning or at the end the list not in the middle. Two of the Data Structures that are useful in such situations are stacks and queues. A stack is a list of elements in which an element may be inserted or deleted only at one end, called the top. This means, in particular, the elements are removed from a stack in the reverse order of that which they are inserted in to the stack. The stack also called "list-in first-out (LIFO)" list.

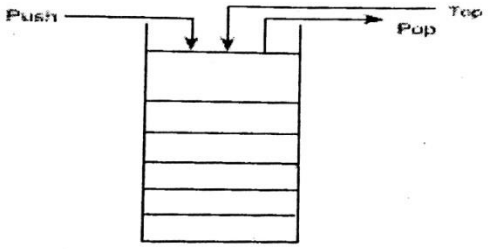


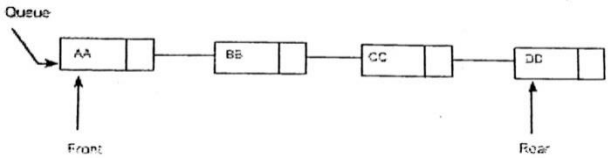
Fig: Stack

Specially terminology is used for two basic operation associated with stack:

1. "Push" is the term used to insert an element into a stack
2. "Pop" is the term used to delete an element from a stack

✓ **Queue:** A queue is a linear list of elements in which deletions can take place only at one end, called the front and insertion can take place only at the other end, called rear. The term front and rear are used in describing a linear list only when it is implemented as a queue.

Queue are also called "first-in first-out (FIFO) list. Since the first element in queue will be the first element out of the queue. In other words, the order in which elements enter in a queue is the order in which they leave. The real life example: the people waiting in a line at Railway ticket Counter form a queue, where the first person in line is the first person to be waited on. An important example of a queue in computer science occurs in timesharing system in which programs with the same priority from the queue while waiting to be executed.



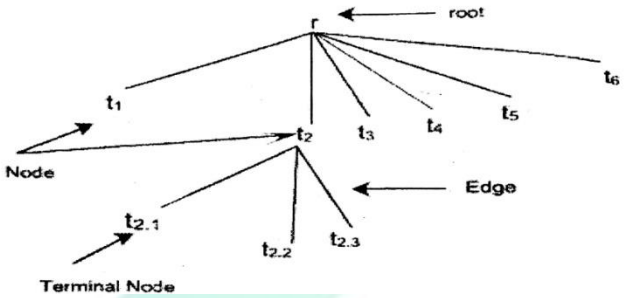
Non - Linear Data Structure: Non linear data structure is the hierarchical relationship between individual data items. Every data item is attached to several other data items in a way that is specific for reflecting relationship. The data items are not arranged in a sequential structure. Ex: Tree, Graphs

✓ **Tree:** Data frequently contain a hierarchical relationship between various elements. This non-linear Data structure which reflects this relationship is called a rooted tree

graph or, tree. This structure is mainly used to represent data containing a hierarchical relationship between elements, e.g. record, family tree and table of contents

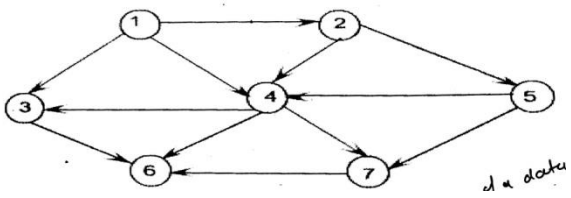
A tree consists of a distinguished node r, called the root and zero or more (sub) tree t1, t2, …. tn, each of whose roots are connected by a directed edge to r.

In the tree of below figure, the root is r, Node t_2 has r as a parent and $t_{2.1}$, $t_{2.2}$ and $t_{2.3}$ as children. Each node may have arbitrary number of children, possibly zero. Nodes with no children are known as leaves.



✓ **Graph:** A graph consists of a set of Vertices (nodes) and a set of edges. Each edge in a graph is specified by a pair of vertices. A vertex n is incident to an edge x if n is one of the two nodes in the ordered pair of nodes that constitute x. The degree of a node is the number o arcs incident to it. The indegree of a node n is the number of arcs that have n as the head, and the outdegree of n is the number of arcs that have n as the tail.

The graph is the nonlinear data structure. The graph show in below figure represents 7 vertices and 12 edges. The Vertices are {1, 2, 3, 4, 5, 6, 7} and the arcs are {(1,2), (1,3), (1,4), (2,4), (2,5), (3,4), (3,6), (4,5), (4,6), (4,7), (5,7), (6,7)}. Node (4) in below figure has indegree 3, outdegree 6.



ABSTRACT DATA TYPE

Abstract Data Types (ADTs) are a model used to understand the design of view of the data structure. ADTs specify the type of data stored and the operations that support the

data. Viewing a data structure as an ADT allows a programmer to focus on a idealized model of the data and its operations.

We can think of an ADT as a mathematical model with a collection of operations defined on that model. Sets of integers, together with the operations of union, intersection, and set difference, form a simple example of an ADT.

DATA TYPE, DATA STRUCUTRE AND ABSTRACT DATA TYPES

Although the terms “data type”, “data structure” and “abstract data type” sounds alike, they have different meanings. In a programming language, the data type of a variable is the set of values that the variable may assume. For example, a variable of type boolean can assume either the value true or the value false, but no other value. The basic data types vary from language to language; in ‘C’ they are integer, float, and char. The rules for constructing data types out of basic ones also vary from language to language.

An abstract data type is a mathematical model, together with various operations defined on the model. We shall design algorithms in terms of ADTs, but to implement an algorithm in a given programming language we must find some way of representing the ADT’s in terms of the operators and data types supported by the programming language itself. To represent the mathematical model underlying ADT we use data structures, which are collections of variables, possibly of several different data types, connected in various ways.

OPERATIONS ON DATA STRUCTURE

The data appearing in out data structure is processed by means of certain operations. The particular data structure that one chooses for a given situation depends largely on the frequency with which specific operations are performed. The following four operations play a major role:

- ✓ **Traversing:** Traversing means accessing each record exactly once so that certain items in the record may be processed. (This processing or accessing is sometimes called visiting the records.)
- ✓ **Searching:** It means finding the location of the record with a given key value, or finding the locations of all record, which satisfy one or more conditions.
- ✓ **Inserting:** Adding new records to the structure.
- ✓ **Deleting:** Removing a record from the structure.

There are two more operations, which can be used in special situations and are discussed below:

- ✓ **Sorting:** Arranging the records in some logical order (e.g., alphabetically in ascending or descending order according to name, or in numerical order according to some number key, etc.)
- ✓ **Merging:** Combining the records in two different sorted files into single sorted file.

Some other operations such as copying and concatenation may also be performed on the some data structures.

ARRAYS

INTRODUCTION

One of the basic and important data structures of a program is Array. It is a data structure which can represent a collection of elements of same data type. An array can be of any dimensions. It can be one, two or multidimensional. An array is generally used when a large amount of data is to be stored.

The simplest form of array is a one - dimensional(1-D) array that may be defined as a finite ordered set of homogeneous elements, which is stored in contiguous memory locations. For example, an array may contain all integers or all characters or any other data type, but may not contain a mix of different data types. Various operations

such as traversal, insertion, deletion, searching and sorting can be performed on arrays.

PRESENTATIONS OF CONTENTS

Linear Arrays

It is probably the most widely used data structure; in some languages it is even the only one available. An array consists of components which are of the same type; it is therefore called a homogeneous structure. It is a random - access structure, because all components can be selected at random and are equally quickly accessible. In order to denote individual component, the name of the entire structure is augmented by the index selecting the component. This index is generally an integer between 0 and n - 1, where n is the number of elements, the size, of the array.

Individual component of an array can be selected by an index. Given an array variable x, we can denote an array selector by the array name followed by the respective component's index i, and can be written as x_i or x[i]. Because of the first, conventional notional, a component of an array component is therefore also called a subscripted variable.

The general equation that can be used to find the length or number of data elements of the array is:

$$\text{Length} = UB - LB + 1$$

Where, UB is the largest index, called the upper bound, the LB is the smallest index, called the lower bound of the array which is generally zero or one. Therefore when LB = 0, Length (Array) = UB - 1 and the length = UB when LB = 1

For example, Consider an Array A as a 6 - element linear array integers such that:

A[1] = 247, A[2] = 56, A[3] = 429. A[4] = 135, A[5] = 87, A[6] = 156. Here UB = 7 and LB = 6. Therefore, length [A] = 6

The following figures can be used to represent the array A.

247
56
429
135
87
156

Figure 1

247	56	429	135	87	156

Figure 2

Declaring an array: The general form for declaring a single dimensional array is:

data_type array_name [expression];

where data_type represents data type of the array i.e., integer, chart, float etc., array_name is the name of array and expression which indicates the maximum number of elements in the array.

For example, consider the following C declaration:

int a [100];

It declares an array of 100 integers.

The mount of storage required to hold an array is directly related to its type of size. For a single dimension (1-D) array, the total size in bytes required for the array is computed as shown below.

Memory required (in bytes) = size of (data type) X length of array

The first array index value referred to as its lower bound and in C programming it is always 0 and the maximum index value is called its upper bound. The number of elements in the array, called its range is given by upper bond - lower bound + 1.

We store values in the arrays during execution of program. Let us see the process of initializing an array while declaring it.

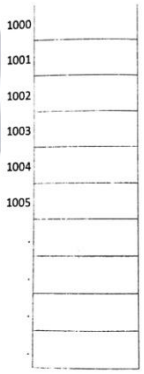
```
int a[4] = {34, 60, 93, 2};

int b[ ] = {2, 3, 4, 5};

float c [ ] = { - 4, 6 81,    60};
```

We conclude the following facts from these examples while using C:

a. If the array is initialized at the time of declaration, then the dimension of the array is optional
b. Till the array elements are not given any specific values, then they contain garbage values.

Representation of Linear Arrays in Computer Memory

The linear arrays in computer memory are represented using contiguous memory locations. For example, consider A to be a linear array in the computer memory. As we know that the computer memory is simply a sequence of addressed location as shown in below figure.



Then we can use the following notation for calculating the address of any element in linear arrays in computer memory.

LOC (A[K]) = address of the element A[K] of the array A.

Here the computer does not need to keep track of the address of every element of array A, but only needs to keep track of the address of the first element of A, which is denoted by Base (A) and termed as the base address of array A.

The computer calculates the address of any element of an array A using the following formula:

$$LOC(A[K]) = Base(A) + W*(K - LB)$$

Where W is the number of words per memory cell for the array A e.g., in case of float in C the value of W is 4.

For example, consider an array A, which records the sales of a company in each year from 1832 through 1884.

Assume, Base (A) = 100 and W = 4 words per memory cell.

Then the base addresses of following arrays are,

LOC(A[1832]) = 100, LOC(A[1833]) = 104, LOC(A[1834]) = 108, ...

Let us find the address of the array element for the year K = 1875. It can be obtained in this way:

LOC(A[1875]) = Base(A) + W*(1875 - LB)

LOC(A[1875]) = 100 + 4*(1875 - 1832) = 272

Again, it is clear that the contents of this element can be obtained without scanning any other element in array A.

Operations on Arrays

We can perform the following operations on arrays

a. **Traversal:** Processing each element in the arrays
b. **Search:** It means finding the location of the element with a given value
c. **Insertion:** Adding a new element to the array.
d. **Deletion:** Removing an element from the array
e. **Sorting:** Arranging the elements in some type of order i.e. or descending
f. **Merging:** Combining two arrays into a single array.

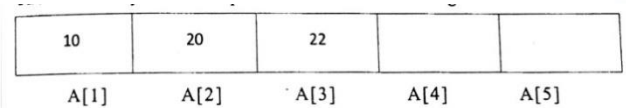
Inserting and Deleting

Let A be an array stored in the memory of the computer. Inserting means the operation of adding another element to the array A, and deleting means the operation of removing one of the elements from array A. let us discuss the procedure of inserting and deleting an element when A is a linear array.

Inserting the element at the end of a linear array can be easily done provided the memory space allocated for the array is large enough to accommodate the additional element. On the other hand, suppose we need to insert an element in the middle of the array. Then, on the average, half of the elements must be moved downward to new locations to accommodate the new element and keep the order of the other elements.

Similarly, deleting the element at the end of an array present no difficulties, but deleting an element somewhere in the middle of an array would require that each subsequent element be moved one location upward in order to fill up the empty space in the array.

For example consider an array A has been declared as a 5 - element array but data have been recorded only for A[1], A[2], and A[3]. Then the array can be represented as shown in below figure



If X is the value to the next element, the new may simply assign, A[4]: = X to add X to the Linear Array. Similar, if Y is the value of the subsequent element, then we may assign, A[5]: = Y to add Y to the Linear Array. Now, we may conclude that we cannot add any new element to this Linear Array due to the reach of upper bound.

Now consider another example, suppose MARKS is an 8 - element linear array, and suppose five marks are in the array, as in below figure. Assume that the marks are stored in ascending order, and suppose we want to keep the array sorted at all

times. Then, if e want to insert a new element, we may have to move the data downwards as shown in below figure and if we want to delete some data, we may have to move the data upwards.

10	20	30	40				

Suppose 25 needs to be inserted in the above array, then the new array would be

10	20	25	30	40			

Now assume that 20 needs to be deleted form the above array, then the new array would be

10	25	30	40			

It can be observed clearly that such movement of data would be very expensive if thousands of data items are in the array.

Multidimensional Arrays

The linear arrays discussed so far are also called one dimensional array, since each element in the array is referenced by a single subscript. Now we will be discussion the multidimensional array, i.e., each element of the array will now be referenced by more than one subscript depending on the number of dimensions. Most programming languages allow two - dimensional or three - dimensional arrays in general, but some programming languages allow the number of dimensions in an array to be higher.

Two - Dimensional Arrays

A two - dimensional m x n array A is a collection of m.n data elements such that each element is specified by a pair of integers (such as I, J), called subscripts, with the following property that,

$$1 <= I <= m \text{ and } 1 <= J <= n$$

The element of A with first subscripts i and second subscript j will be denoted by $A_{1,J}$ or A[I, J]

Two - dimensional arrays are generally called matrices in mathematics and tables in business applications; hence two - dimensional arrays are called matrix arrays.

The schematic of a two - dimensional array of size 3 x 5 is shown in below figure

A[0][0]	A[0][1]	A[0][2]	A[0][3]	A[0][4]
ROW – 1				
A[1][0]	A[1][1]	A[1][2]	A[1][3]	A[1][4]
ROW – 2				
A[2][0]	A[2][1]	A[2][2]	A[2][3]	A[2][4]
ROW – 3				

For example the scores of five player in five matches can be shown in below figure

Player	Match 1	Match 2	Match 3	Match 4	Match 5
A	29	78	55	0	4
B	65	101	88	9	76
C	0	0	76	199	88
D	9	8	76	44	65
E	44	5	3	9	100

The above figure shows a 5*5 array consisting of 25 elements.

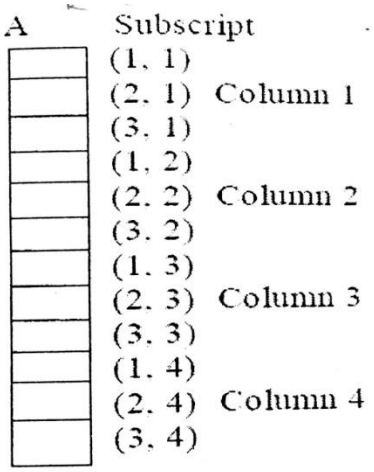
In the case of a two – dimensional(2-D) array, the following formula yields the number of bytes of memory needed to hold it:

Bytes = size of 1st index X size of 2nd index X size of (base type)

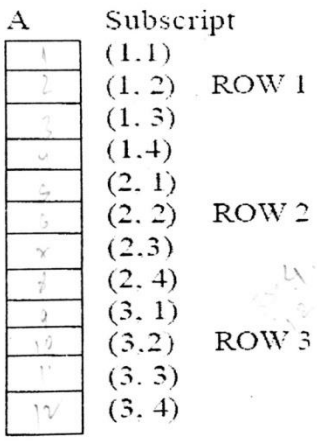
Representation of two – Dimensional Arrays in Memory

Let A be a two – dimensional(2-D) m X n array. Although A is views as a rectangular array of elements with m rows and n columns, the array will be represented in memory by a block of m*n sequential memory locations. Now questions arises that how these elements are sequenced i.e. whether the elements are stored row wise or column wise? It actually depends on the operating system. Specifically, the programming languages will store the array A in either, column by column, called – major order, or, row by row called row – major order.

The figure below shows these two ways when A is a two – dimensional 3 x 4 arrays.



Column-major order.



Row-major order. Fi

As we able to find address of any element of the array in case of linear array, a similar situation also holds for any two – dimensional M x N array A i.e., the computer keeps tracks of base (A), which is the address of the first element A[1, 1] of A and computers the address LOC(A[I, J]).

The formula for column major order is,

$$LOC(A[I, J]) = Base (A) + W*[M(J-1) + (I – 1)]$$

The formula for row major order is

$$LOC(A[I, J]) = Base (A) + W*[M(I-1) + (J – 1)]$$

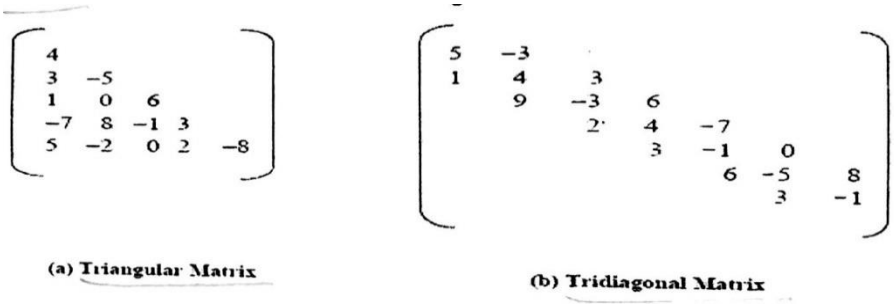
For example, consider the previous example of above figure, of 5 X 5 matrix array SCORE. Suppose Base (SCORE) = 200 and there are W = 4 words per memory cell and let the programming language stores two – dimensional arrays using row – major order. Then the address of SCORE [3,3], the third match of the third player follows:

$$LOC(SCORE[3, 3]) = 200 + 4*[5*(3 – 1) + (3 – 1)] = 200 + 4*[12] = 248$$

Two – dimensional arrays clearly shows the difference between the logical and the physical views of data. Above figures shows that how we could logically views a 5 X 5 matrix array A. On the other hand, the data will be physically stored in memory by a linear collection of memory cells.

Spares Matrices

Matrices with good number of zero entries are called sparse matrices. Consider the matrices of below figure.



(a) Triangular Matrix

(b) Tridiagonal Matrix

A triangular matrix is a square matrix in which all the elements either above or below the main diagonal are zero. Triangular matrices are sparse matrices. A tridiagonal matrix is a square matrix in which all the elements except for the main diagonal, diagonals on the immediate upper and lower side are zeros. Tridiagonal matrices are also sparse matrices

APPLICATIONS OF ARRAYS

Arrays are simple, but reliable to use in more situations than you can count. Arrays are used in those problems when the number of items to be solved is fixed. They are easy to traverse, search and sort. It is very easy to manipulate an array rather than other subsequent data structures. Arrays are used in those situations where in the size of array can be established beforehand. Also, they are used in situations where the insertions and deletions are minimal or not present. Insertion and deletion operations will lead to wastage of memory or will increase the time complexity of the program due to the reshuffling of elements.

gradeup SUPER

Gradeup UGC NET

Super Superscription

Features:

1. 7+ Structured Courses for UGC NET Exam
2. 200+ Mock Tests for UGC NET & MHSET Exams
3. Separate Batches in Hindi & English
4. Mock Tests are available in Hindi & English
5. Available on Mobile & Desktop

Gradeup Super Subscription, Enroll Now

gradeup.co