CLASSIFICATION OF DATA STRUCTURE

Data structures are broadly divided into two:

1. **Primitive data structures**: These are the basic data structures and are directly operated upon by the machine instructions, which is in a primitive level. They are integers, floating point numbers, characters, string constants, pointers etc. These primitive data structures are the basis for the discussion of more sophisticated (non-primitive) data structures in this book.

2. **Non-primitive data structures**: It is a more sophisticated data structure emphasizing on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items. Array, list, files, linked list, trees and graphs fall in this category.

![Diagram of Data Structure分类](image)

The Fig. 1.4 will briefly explain other classifications of data structures. Basic operations on data structure are to create a (non-primitive) data structure; which is considered to be the first step of writing a program. For example, in Pascal, C and C++, variables are created by using declaration statements.

\[ \text{int Int_Variable;} \]

In C/C++, memory space is allocated for the variable "Int_Variable" when the above declaration statement executes. That is a data structure is created. Discussions on primitive data structures are beyond the scope of this book. Let us consider non-primitive data structures.

```markdown
Data Structure

<table>
<thead>
<tr>
<th>Primitive Data Structure</th>
<th>Non-Primitive Data Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Float</td>
</tr>
<tr>
<td>Arrays</td>
<td>Lists</td>
</tr>
<tr>
<td>Linear Lists</td>
<td>Non-Linear Lists</td>
</tr>
<tr>
<td>Stacks</td>
<td>Queues</td>
</tr>
</tbody>
</table>
```

![Diagram of Data Structure](image)

Fig. 1.4. Classifications of data structures
ARRAYS

Arrays are most frequently used in programming. Mathematical problems like matrix, algebra and etc can be easily handled by arrays. An array is a collection of homogeneous data elements described by a single name. Each element of an array is referenced by a subscripted variable or value, called subscript or index enclosed in parenthesis. If an element of an array is referenced by single subscript, then the array is known as one dimensional array or linear array and if two subscripts are required to reference an element, the array is known as two dimensional array and so on. Analogously the arrays whose elements are referenced by two or more subscripts are called multi dimensional arrays.

ONE DIMENSIONAL ARRAY

One-dimensional array (or linear array) is a set of ‘n’ finite numbers of homogenous data elements such as:

1. The elements of the array are referenced respectively by an index set consisting of ‘n’ consecutive numbers.
2. The elements of the array are stored respectively in successive memory locations.

‘n’ number of elements is called the length or size of an array. The elements of an array ‘A’ may be denoted in C as

\[ A[0], A[1], A[2], \ldots, A[n-1]. \]

The number ‘n’ in A[n] is called a subscript or an index and A[n] is called a subscripted variable. If ‘n’ is 10, then the array elements A[0], A[1],…,A[9] are stored in sequential memory locations as follows:

\[
\begin{array}{cccccc}
\end{array}
\]

In C, array can always be read or written through loop. To read a one-dimensional array, it requires one loop for reading and writing the array, for example:

For reading an array of ‘n’ elements

\[
\text{for (i = 0; i < n; i++)} \\
\text{scanf ("%d", &A[i]);}
\]

For writing an array

\[
\text{for (i = 0; i < n; i++)} \\
\text{printf ("%d", &A[i]);}
\]

MULTI DIMENSIONAL ARRAY

If we are reading or writing two-dimensional array, two loops are required. Similarly the array of ‘n’ dimensions would require ‘n’ loops. The structure of the two dimensional array is illustrated in the following figure:

\[
\begin{array}{cccc}
A_{00} & A_{01} & A_{02} & \ldots & A_{08} & A_{09} \\
A_{10} & A_{11} & \ldots & \vdots & \vdots & A_{19} \\
A_{20} & \ldots & \ddots & \vdots & \vdots \\
A_{30} & \ldots & \ldots & \ddots & \vdots \\
\vdots & \vdots & \ldots & \ddots & \ddots & \ldots \\
A_{70} & \ldots & \ldots & \ldots & \ddots & A_{78} & A_{79} \\
A_{80} & A_{81} & \ldots & \ldots & \ldots & A_{87} & A_{88} & A_{89} \\
\end{array}
\]
Sparse Arrays

Sparse array is an important application of arrays. A sparse array is an array where nearly all of the elements have the same value (usually zero) and this value is a constant. One-dimensional sparse array is called sparse vectors and two-dimensional sparse arrays are called sparse matrices.

The main objective of using arrays is to minimize the memory space requirement and to improve the execution speed of a program. This can be achieved by allocating memory space for only non-zero elements.

For example a sparse array can be viewed as

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 9 & 0 \\
0 & 0 & 0 & 3 & 0 & 0 & 0 \\
0 & 31 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 0 & 7 & 0 & 0 \\
\end{array}
\]

**Fig. 1.5.** Sparse array

We will store only non-zero elements in the above sparse matrix because storing all the elements of the sparse array will be consisting of memory sparse. The non-zero elements are stored in an array of the form

\[A[0......n][1......3]\]

Where ‘n’ is the number of non-zero elements in the array. In the above Fig. 1.4 ‘n = 7’. The space array given in Fig. 1.4 may be represented in the array A[0......7][1......3].

**Fig. 1.6.** Sparse array representation

The element A[0][1] and A[0][2] contain the number of rows and columns of the sparse array. A[0][3] contains the total number of nonzero elements in the sparse array.
A[1][1] contains the number of the row where the first nonzero element is present in the sparse array. A[1][2] contains the number of the column of the corresponding nonzero element. A[1][3] contains the value of the nonzero element. In the Fig. 1.4, the first nonzero element can be found at 1st row in 3rd column.

VECTORS

A vector is a one-dimensional ordered collection of numbers. Normally, a number of contiguous memory locations are sequentially allocated to the vector. A vector size is fixed and, therefore, requires a fixed number of memory locations. A vector can be a column vector which represents a ‘n’ by 1 ordered collections, or a row vector which represents a 1 by ‘n’ ordered collections.

The column vector appears symbolically as follows:

\[
A = \begin{pmatrix}
A_1 \\
A_2 \\
A_3 \\
\vdots \\
A_n
\end{pmatrix}
\]

A row vector appears symbolically as follows:

\[
A = (A_1, A_2, A_3, \ldots, A_n)
\]

Vectors can contain either real or complex numbers. When they contain real numbers, they are sometime called real vectors. When they contain complex numbers, they are called complex vectors.

LISTS

As we have discussed, an array is an ordered set, which consist of a fixed number of elements. No deletion or insertion operations are performed on arrays. Another main disadvantage is its fixed length; we cannot add elements to the array. Lists overcome all the above limitations. A list is an ordered set consisting of a varying number of elements to which insertion and deletion can be made. A list represented by displaying the relationship between the adjacent elements is said to be a linear list. Any other list is said to be non linear. List can be implemented by using pointers. Each element is referred to as nodes; therefore a list can be defined as a collection of nodes as shown below:

![Fig. 1.7](mywbut.com)