Chapter 1

Data Structures

Data structure is a way to organize a data in computer so that it can be used efficiently.

How are data structures used?

Data structures handle four main functions for us:

- Inputting information:

  Inputting is largely concerned with how the data is received. What kind of information can be included? Will the new data be added to the beginning, end, or somewhere in the middle of the existing data? Does an existing point of data need to be updated or destroyed?

- Processing information

  Processing gets at the way that data is manipulated in the data structure. This can occur concurrently or as a result of other processes that data structures handle. How does existing data that has been stored need to change to accommodate new, updated, or removed data?

- Maintaining information

  Maintaining is focused on how the data is organized within the structure. Which relationships need to be maintained between pieces of data? How much memory must the system reserve (*allocate*) to accommodate the data?

- Retrieving information

  Retrieving is devoted to finding and returning the data that is stored in the structure. How can we access that information again? What steps does the data structure need to take to get the information back to us?

Need for Data Structure

As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

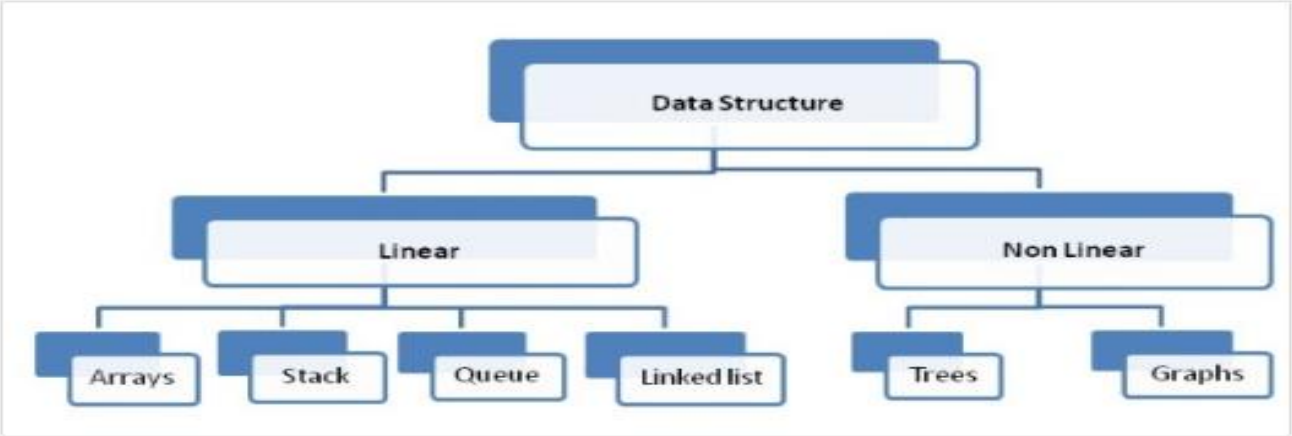
- Data Search − Consider an inventory of 1 million(106) items of a store. If the application is to search an item, it has to search an item in 1 million(106) items every time slowing down the search. As data grows, search will become slower.

- Processor speed − Processor speed although being very high, falls limited if the data grows to billion records.
- Multiple requests − As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

To solve the above-mentioned problems, data structures come to rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

Data Structure is classified into two categories :

1. Linear Data Structure
2. Non Linear Data Structure



Classifications of Data Structure

Linear Data Structures:

The data structure where data items are organized sequentially or linearly where data elements attached one after another is called linear data structure. Data elements in a liner data structure are traversed one after the other and only one element can be directly reached while traversing. All the data items in linear data structure can be traversed in single run.

- Various operations on a data structure are possible only in a sequence i.e. we cannot insert the element into any location of our choice. E.g. A new element in a queue can come only at the end, not anywhere else.
- It consumes linear memory space, the data elements are required to store in a sequential manner in the memory.
- It does not make a good utilization of memory and result in memory wastage.
- While implementing the linear data structure the necessary amount of memory is declared previously
- They can be implemented in memory using two ways.

i. By having a linear relationship between elements by means of sequential memory locations.
ii. By having a linear relationship by using links.

- Examples of linear data structures are **Arrays, Stack, Queue** and **Linked List.**

Non Linear Data Structures:

The data structure where data items are not organized sequentially is called nonlinear data structure. In other words, A data elements of the nonlinear data structure could be connected to more than one elements to reflect a special relationship among them. All the data elements in nonlinear data structure cannot be traversed in single run.

- When the data elements are organised in some arbitrary function without any sequence, such data structures are called non-linear data structures.
- The relationship of adjacency is not maintained between elements of a non-linear data structure.
- In this, the data elements can be attached to more than one element exhibiting the hierarchical relationship which involves the relationship between the child, parent, and grandparent.
- In the non-linear data structure, the traversal of data elements and insertion or deletion is not done sequentially.
- Examples of nonlinear data structures are **Trees** and **Graphs**.

Difference Between Linear and Non Linear Data Structure

BASIS FOR COMPARISON	LINEAR DATA STRUCTURE	NON-LINEAR DATA STRUCTURE
• Basic	• The data items are arranged in an orderly manner where the elements are attached adjacently.	• It arranges the data in a sorted order and there exists a relationship between the data elements.
• Traversing of the data	• The data elements can be accessed in one time (single run).	• Traversing of data elements in one go is not possible.
• Ease of implementation	• Simpler	• Complex

BASIS FOR COMPARISON	LINEAR DATA STRUCTURE	NON-LINEAR DATA STRUCTURE
• Levels involved	• Single level	• Multiple level
• Memory utilization	• Ineffective	• Effective
• Examples	• Array, queue, stack, linked list, etc.	• Tree and graph.

Algorithmic Complexity

Data structures are implemented using algorithms. An algorithm is a procedure that you can write as a C function or program, or any other language. An algorithm states explicitly how the data will be manipulated.

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

- **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
- **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm f(n) gives the running time and/or the storage space required by the algorithm in terms of n as the size of input data.

Space Complexity

- Space complexity is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm. We often speak of "extra" memory needed, not counting the memory needed to store the input itself.
- Again, we use natural (but fixed-length) units to measure this. We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit. Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.
- Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

- A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
- A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I. Following is a simple example that tries to explain the concept −

```
Algorithm: SUM(A, B)
Step 1 -  START
Step 2 -  C ← A + B + 10
Step 3 -  Stop
```

Here we have three variables A, B, and C and one constant. Hence S(P) = 1 + 3. Now, space depends on data types of given variables and constant types and it will be multiplied accordingly.

Time Complexity

Time complexity of an algorithm signifies the total time required by the program to run till its completion.

Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

Calculating Time Complexity

Now the most common metric for calculating time complexity is Big O notation. This removes all constant factors so that the running time can be estimated in relation to N, as N approaches infinity. In general you can think of it like this :

1. statement;

Above we have a single statement. Its Time Complexity will be Constant. The running time of the statement will not change in relation to N.

2. for(i=0; i < N; i++)
   {
      statement;
   }

The time complexity for the above algorithm will be Linear. The running time of the loop is directly proportional to N. When N doubles, so does the running time.

3. for(i=0; i < N; i++)
   {

```
for(j=0; j < N;j++)
  {
    statement;
   }
  }
```

This time, the time complexity for the above code will be Quadratic. The running time of the two loops is proportional to the square of N. When N doubles, the running time increases by N * N.

- "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.
- We try to keep this idea of time separate from "wall clock" time, since many factors unrelated to the algorithm itself can affect the real time (like the language used, type of computing hardware, proficiency of the programmer, optimization in the compiler, etc.).
- It turns out that, if we chose the units wisely, all of the other stuff doesn't matter and we can get an independent measure of the efficiency of the algorithm.

Operations on Data Structures

The basic operations that are performed on data structures are as follows:

1. Insertion: Insertion means addition of a new data element in a data structure.
2. Deletion: Deletion means removal of a data element from a data structure if it is found.
3. Searching: Searching involves searching for the specified data element in a data structure.
4. Traversal: Traversal of a data structure means processing all the data elements present in it.
5. Sorting: Arranging data elements of a data structure in a specified order is called sorting.
6. Merging: Combining elements of two similar data structures to form a new data structure of the same type, is called merging.