**Course: C339 Data Fundamentals**

**Date: February 28, 2023**

**Title: Loops and Data Structures**

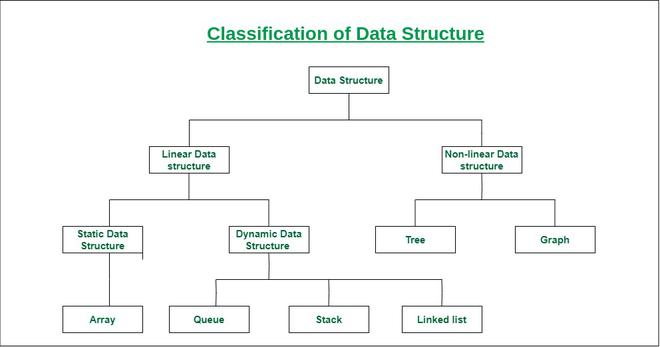
****

Figure 1 - Classification of data structure.

**Linear vs. non-linear data structures**

The Difference Between Linear And Non-Linear Data Structures is that Linear data structures are arranged sequentially and linearly. In contrast, the non-linear data structure is where data is not arranged sequentially and linearly. Data structures are the method to store and organize the information/data for the convenience of the user.

The major difference between linear and non-linear data structures is that linear data structures are ordered. In contrast, non-linear state structures are not ordered and are distributed in a random manner.

Key differences between linear and non-linear data structures:

| **Linear Data structures** | **Non-linear data structures** |
| --- | --- |
| Elements are ordered in a linear and sequential manner. | Elements are ordered in a hierarchy |
| ONly a single level is present. | Multiple level data structures are present |
| Implementation is relatively easier. | Implementation is relatively complicated. |
| Traversed in a single run. | Take multiple runs to traverse the data. |
| Memory utilisation is not efficient compared to non-linear data structures | Memory utilisation is efficient. |
| Examples: Array, queue, stack, linked list, hash tables, etc. | Examples: Trees, graphs, etc. |

**Linear data structures**

Linear data structures are structures where data elements are ordered in a sequential or linear way. In the linear data structure, each element is attached to the elements before and after them. The involvement of the structures is at a single level only. They are easy to implement.

| **Data type** | **Explanation** |
| --- | --- |
| Array | An array is a collection of items of same data type stored at contiguous memory locations.  This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array). The base value is index 0 and the difference between the two indexes is the offset. |
| Queue | A queue is a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that. Thus, the operations are performed in First In First Out (FIFO) order. |
| Stack | A stack follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). LIFO implies that the element that is inserted last, comes out first and FILO implies that the element that is inserted first, comes out last. |
| Linked list | A linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. |
| Dictionary | The dictionary Data Structure in Python is an unordered collection of items. While other Data Structures use only one value as the element, the dictionary is a slightly more compound data structure. It makes use of two elements i.e. a pair of elements, namely, a key and a value.  Dictionary values can be accessed with their respective keys. To access a specific value in the dictionary data set, you need to index the right key. Dictionaries in Python are mutable and the elements in a dictionary can be added, removed, modified, and changed accordingly. |

**Non-linear data structures**

Non-linear data structures are structures where data is not ordered in a sequential or linear way. There exist multiple levels of arrangement. It requires multiple run-downs to traverse the elements. The elements may or may not be attached to them before and after. They are known to be more efficient than linear data structures.

| **Data type** | **Explanation** |
| --- | --- |
| Trees | Tree is an abstract data type that represents a hierarchical tree structure with a set of connected nodes.  Basic Terminologies In Tree Data Structure:  **Parent Node:** The node which is a predecessor of a node is called the parent node of that node. {B} is the parent node of {D, E}.  **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: {D, E} are the child nodes of {B}.  **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {A} is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.  **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. {K, L, M, N, O, P} are the leaf nodes of the tree.  **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node. {A,B} are the ancestor nodes of the node {E}  **Descendant:** Any successor node on the path from the leaf node to that node. {E,I} are the descendants of the node {B}.  **Sibling:** Children of the same parent node are called siblings. {D,E} are called siblings.  **Level of a node:** The count of edges on the path from the root node to that node. The root node has level 0.  **Internal node:** A node with at least one child is called Internal Node.  **Neighbour of a Node:** Parent or child nodes of that node are called neighbors of that node.  **Subtree:** Any node of the tree along with its descendant. |
| Graphs | Graphs are non-linear data structures made up of two major components:  **Vertices –** Vertices are entities in a graph. Every vertex has a value associated with it. For example, if we represent a list of cities using a graph, the vertices would represent the cities.  **Edges –** Edges represent the relationship between the vertices in the graph. Edges may or may not have a value associated with them. For example, if we represent a list of cities using a graph, the edges would represent the path between the cities. |

**Prefix arrays**

Prefix arrays help to minimize the repeated calculation done in an array and thus reduces the time complexity of your program.

| **Index (i)** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Array A [A]** | 6 | 3 | -2 | 4 | -1 | 0 | -5 |
| **Prefix sum array [PA]** | **6** | **9** | **7** | **11** | **10** | **10** | **5** |
|  | 0+6 | 6+3 | 9+(-2) | 7+4 | 11+(-1) | 10+0 | 10+(-5) |

Table 1 - Indexes, Arrays, and prefix sum array.

The same result can be achieved by sequentially adding the array elements:

| Index (i) | Prefix sum array [PA] | Sequential addition |  |
| --- | --- | --- | --- |
| 0 | 6 | 0 + 6 | This process (of sequential addition) can be expressed in a loop:  Thus, it can be said that this loop has a linear complexity O(N). N is the number of times the loop is executed (and this depends on user input/number of arrays). |
| 1 | 9 | 0 + 6 + 3 |
| 2 | 7 | 0 + 6 + 3 + (-2) |
| 3 | 11 | 0 + 6 + 3 + (-2) + 4 |
| 4 | 10 | 0 + 6 + 3 + (-2) + 4 + (-1) |
| 5 | 10 | 0 + 6 + 3 + (-2) + 4 + (-1) + 0 |
| 6 | 5 | 0 + 6 + 3 + (-2) + 4 + (-1) + 0 + (-5) |

Table 2 - Sequential addition of array elements.

Say we want the third (i = 3), fourth (i = 4), and fifth (i = 5) index from Table 1. Using the method from Table 2, we can write it as:

When i = 3: A[0] + A[1] + A[2] + A[3]

When i = 4: A[0] + A[1] + A[2] + A[3] + A[4]

When i = 5: A[0] + A[1] + A[2] + A[3] + A[4] + A[5]

But this is a little inefficient. Why? Say we want the fourth index (i = 4). We shouldn't have to add all four array elements again as we’ve computed the sum of the first three array elements already. A more efficient to way to obtain (i = 4) would be:

Sum [3] + A[4]

Therefore, it would just be 11 + (-1) compared to 0 + 6 + 3 + (-2) + 4 + (-1). From this, we can define a formula to calculate prefix sum arrays as:

**PA[ i ] = A[ i ] + PA[ i - 1 ] —------- (Formula 1)**

**Question 1 :** Using Table 1, calculate the sum between range [0,4].

**Answer 1** : 10 (A[4])

**Question 2** : Using Table 1, calculate the sum between range [0,6].

**Answer 2** : 5 (A[6])

**Question 3** : Using Table 1, calculate the sum between range [2,6].

**Answer 3** : Sum between range [0,6] = sum between [0,1] + sum between [2,6]

A[6] = A[1] + sum between [2,6]

sum between [2,6] = A[6] - A[1] (see below)

= 5 - 9

= 4

Visualizing Question 3

| **Index (i)** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Array A [A]** | 6 | 3 | -2 | 4 | -1 | 0 | -5 |
|  | A[6] | | | | | | |
| A[1] | | A[2,6] | | | | |
| **Prefix sum array [PA]** | **6** | **9** | **7** | **11** | **10** | **10** | **5** |

Sum between [2,6] = A[6] - A[1] can be written as: A[2,6] = A[6] - A[1]. From this, the formula to calculate the sum between range [ i , j ] is

**A[ i , j ] = A[ j ] - A[ i - 1] —--------- (Formula 2)**

Where i = starting index and j = ending index.

Question 4 : Using Table 1 and Formula 2, calculate the sum between range [3,5].

Answer 4 : A[3,5] = A[5] - A[2]

= 10 - 7

= 3