National Conference on Research Challenges and Opportunities in Digital and Cyber Forensics
International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2020 IJSRCSEIT | Volume 4 | Issue 11 | ISSN : 2456-3307 (www.ijsrcseit.com)

Data Structures - A Comparative Analysis and Application in Cyber Security

K Reshma, Reem Fatima, Ria Mohan

Department of Information Science & Engineering, New Horizon College of Engineering, Outer Ring Road, Marathahalli, Bangaluru, India

ABSTRACT

Data handling in C is a pre-ordained part of programs. Computer programs often process data, so we require competent ways in which we can access or deploy data. In order to do this, we use a structure called “data structure". Data Structure is a data organization, management, and storage system that enables efficient access and amendment. Data Structures being the mainstay of every software, a good expertise over the subject is essential for all software applications under the field of cyber security. Data Structures is about interpreting data elements in terms of some association, for better organization and storage. In this paper, we'll be drawing a contrast of the various data structures that are used in the C programming language and quoting its applications in cyber security. Doing so will give us a perception on how to capitalize on the performance of a program.

Keywords : Data handling, efficient, organized, data organization, cyber security.

I. INTRODUCTION

A Data structure is a data management, organization, and storage format that facilitates efficient access and alteration. More precisely, a data structure is a set of data values, the relationships among them, and the functions that can be applied to the data.

Data structures deals with the study of how the data is organised in the memory, how effectively the data can be retrieved and manipulated and possible ways in which different data items are logically related. They represent the logical relationship that exists between individual elements of data to carry out certain tasks. [1]

Data structures can be categorized into Primitive data structures and non-primitive data structures. Further non primitive data structures can be grouped as linear data structures and non- linear data structures.

Examples of primitive data structure- int, float, char etc. Examples of non-primitive data structure- arrays, structures etc. Examples of linear data structure- arrays, lists, stacks, queues. Examples of non-linear data structure- trees, graphs.

Arrays are type of data structures that can store a fixed size sequential collection of elements of the same type.[2]

Array is used to store similar data items. Instead of declaring individual variables such as num1, num2, num3,…….numN we can declare one array variable with the same numbers and use num[1], num[2], num[3],….num[N] to represent individual variables. A specific element in array is accessed by index.

Declaring arrays:

datatype
arrayName[size];
ex. Int array[100];

Initialising arrays:

Int a[5]={23,54,67,89,43}

Accessing arrays:

An array element can be accessed by indexing array name.

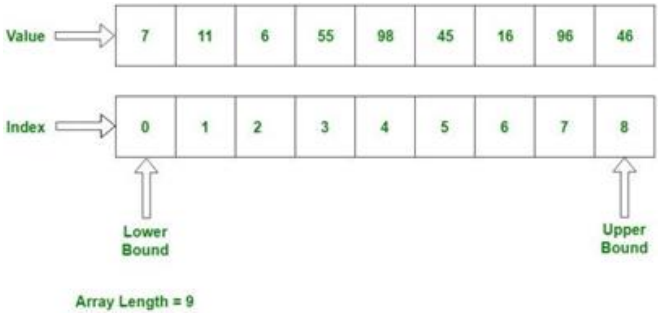

Fig. 1. Representation of an Array

A. *Stack*

A stack is a data structure which stores the elements, retrieves the elements in a sequential way using LIFO (Last In First Out) policy.

LIFO- Last In First Out means that last added element is removed first.

The insertion and deletion operations on stack are done at one end called top. The insertion operation of stack is called push and the deletion operation is called pop.

When an element is inserted into the stack, the overflow condition is checked, when an element is removed from the stack underflow condition is checked. [3]

Overflow:
top==[MAX-1]
Underflow: top==-1

When inserting element in stack, increment top value and insert the element. When removing element from stack, remove top most element and decrement the top index.

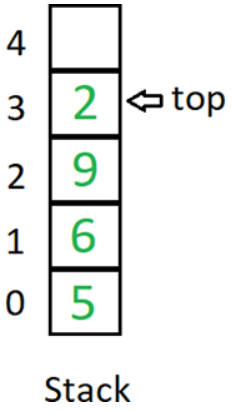

Fig. 2. Representation of a Stack

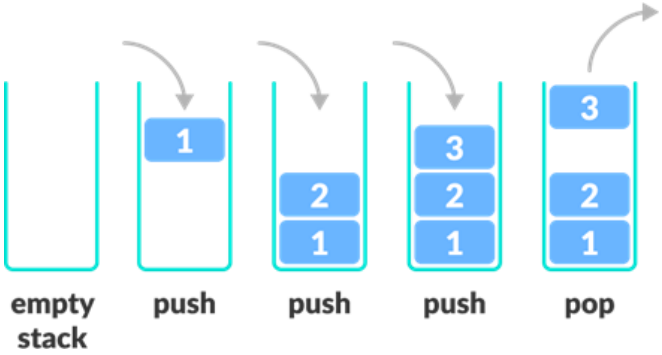

Fig. 3. Stack Operations

B. *Queue*

A queue is a data structure which stores the elements, retrieves the elements in a sequential way using FIFO (First In First Out) policy.

FIFO- First In First Out means that first added element is removed first.

The insertions in a queue are done at the rear end of the queue and the deletions are done at the front end of the queue.

The insertion operation of queue is called en-queue and the deletion operation is called de-queue.

When an element is inserted into the queue, the overflow condition is checked, when an element is removed from the queue underflow condition is checked.[4]

Overflow:
rear==arraysize
Underflow:
front==rear

When inserting element in queue, insert the element to the rear position and then increment rear value. When removing element from queue, initialize the front element to 0 and increment the front value.

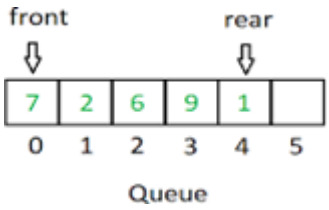

Fig. 4. Representation of a Queue

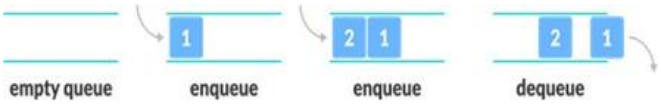

Fig. 5. Queue Operations

D. Linked List

It is a dynamic data structure which consists of non-sequential collection of data items. It is called linear data structure because of its appearance, in which elements are stored at non- contiguous memory locations but are linked to each other using pointers.

Linked list is made up of nodes. Each node consists of a data field and a reference link to the successive node.

Linked list operations involves the traversal of nodes which is done using the temporary variable.[5]

Advantages of linked list over arrays - Dynamic size and ease of insertion/deletion.

Terms involved in linked list:

Link – Each and every link can store some data called an element of the linked list.

Next – Each of the links contain a link to the next link or element called Next.

First – A Linked List as a whole contains the connection link to the first link in the list called First.

Operations supported by a linked list: Insertion – Adds an element to the linked list.

Deletion – Deletes an element from the linked list.

Display – Displays the entire list.

Search – Searches an element in the linked list using the given key.

Delete – Deletes an element in the linked list using the given key or by default.

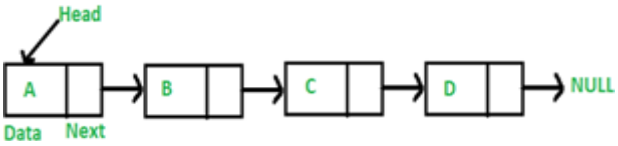

Fig. 6. Representation of a Linked List

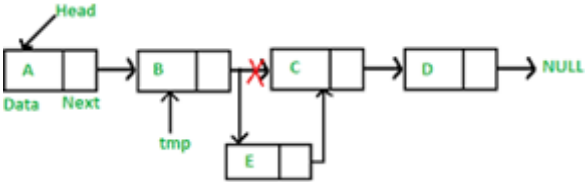

Fig. 7. Linked list Insertion operation

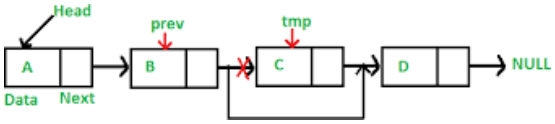

Fig. 8. Linked list Deletion operation

E. Trees

It's a structure that contains nodes which can be connected by edges. Different tree data structures

allow quicker and easier access to the data as it is a non-linear data structure.

The types of trees are Binary search tree(most common), AVl tree and B-tree

In this section, we will be seeing an overview of binary search trees. Its main purpose is for data storage, however binary trees have a unique requirement; each node can only have a total number of two children, and each child can again have only two children and so on. Binary search trees fulfill the purpose of both a sorted array and linked lists.

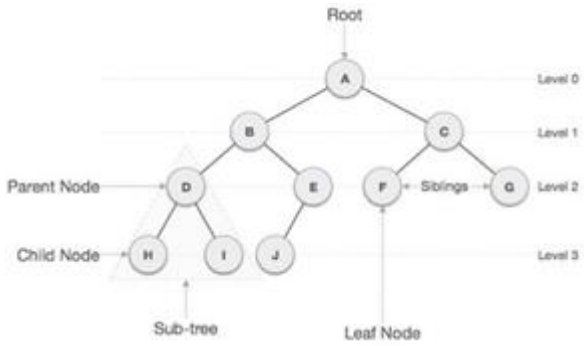

Fig. 9. Representation of a Tree Some terms used in Binary Trees:

Edge: a line connecting two nodes.

Degrees: the complete no. of branches from one node.

Root: the topmost node of tree, and only path from root to other nodes.

Parent: nodes after the root are called parents if they have sub- branches.

Child: each sub node is called a child, and their parent is that node to which they're connected to.

Leaf: if there is a node that has not even one child, it's called leaf node.

Levels: It's to show what generation of parent/child belongs to.

Sub-tree: each segment consisting of a parent and two children apart from the root node is called a sub-tree.

Traversing: methodology for searching a key element in the tree.

Key: the value that the user wishes to find in the tree.

Given a binary tree, there are 3 ways to traverse the tree and find the element the user wishes to find. The 3 operations are called; pre-order, in-order, post-order. After an insertion function is given to take in the input from the user, any of these 3 search operations are executed.

Trees Applications & Advantages:

Time complexity for a non-linear data structure is less.

All the forms of trees are used in a specific environment and it's a customizable data structure.

The stability and reliability that trees lend in terms of security is efficient!

F. Graphs

This data structure is similar to that of trees with certain complexities. Each node contains data and each edge is considered a relationship between the linked data.[6]

For example; On Facebook each element is considered a node and each node is linked, when we post a picture or upload a video onto the platform each element is a node which is

connected to your profile using a link (edge).Thus every time someone establishes a node a new relationship is created.

We can write a graph data structure as an ordered pair of (V,E) where V denotes the collection of all vertices (where the nodes reside containing the data) and E is a collection of edges.

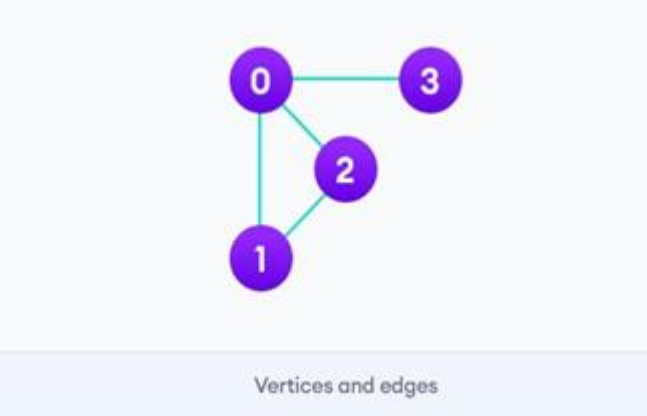

Fig. 10. Representation of a Graph In the above show graph:

V= {0, 1, 2, 3}

E= {(0, 1), (0, 2), (0, 3), (1, 2) G= {V, E}

Graph terminologies:

Adjacency: when there exists an edge that connects two vertices, they're called adjacent vertices. In the above example 2 and 3 are not adjacent.

Path: the designed set of steps that allows us to traverse from one node A to B is a path.

Graph operations:

Function to check if the element is present Traversing the given graph

Function to add a vertex and an edge

Function to display the path between two vertices

Graph Representation:

Adjacency matrix: representation as a 2D matrix where the V*V vertices and in this matrix each row and column is denoting a vertex.

If the value of a[i][j] is returning 1, it implies that at the position i, j there is a data element present. Likewise if it returns 0 then there is no data element present.

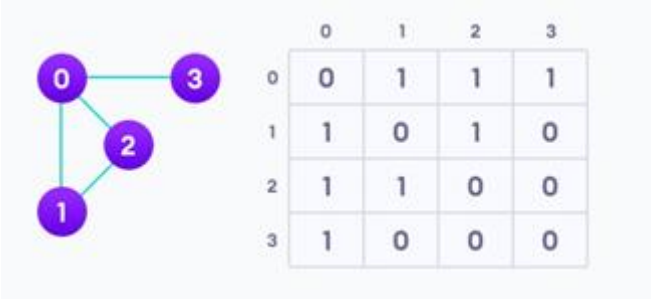

Fig. 11. Adjacency matrix

Adjacency list: representation of the graph as a combined array of linked lists.

The index value of the array will denote the vertex, then each element in the linked list will denote the other vertices that is forming an edge/relationship with the vertex given by the array index.

Fig. 12. Adjacency list

G. Application of Data Structures in Cyber Security

Cryptography is associated with the process of converting typical plain text into indiscernible text and vice-versa. It is a method of accumulating and transmitting data in a particular form so that only those who are authorized and intended to can read and process it. Oblivious data structures, specifically Oblivious Trees, which store relevant data and set of values at its leaves. [9] This property is attained through the use of randomization by the update algorithms.[8] Oblivious tree data structures are in specific used to decipher the privacy problem for incremental digital signatures, as part of cryptography.[10] A

new skill for security has been developed which is a permutation of Caesar Cipher and graph traversal and Binary search tree collectively then security will be much more superior to only using Caesar cipher or graph traversal or binary search tree. [7]

During the process of pen testing or web application scan, one may need to understand the code that's written in order to decipher the particulars.

In many standpoints the need to write appropriate algorithms that gratify the needs within cyber security like code encryption, creating certificates, pen testing arises, and in such cases, having a profound comprehension of data structures plays a very important role.

II. COMPARISON TABLES

Table 1. Comparison of Structured Data

Trees	Graphs
Collection of nodes and edges.	Collection of nodes and edges.
Presence of root nodes.	Root nodes are absent.
No cycle can be formed.	Cycles can be formed.
Only one path exists between two vertices.	Presence of unidirectional and bidirectional paths between vertices.
Trees are simple.	Graphs are complex as it has loops and self-loops.

Table 2. Comparison of Unstructured Data

Array	Stack	Queue	Linked List
Elements belong to indexes.	Follow LIFO policy.	Follow FIFO policy.	Contains collection of unordered linked elements called nodes.
Insertion and deletion can be done at any index.	Insertion and deletion can be done only from top.	Insertion and deletion can be done only from rear and front respectively.	Insertion and deletion can be done at any position.
Dynamic and fixed size.	Dynamic and fixed size.	Dynamic and fixed size.	Dynamic and flexible.
Elements can be of different data types.	Elements should be of same data types.	Elements can be of different data types.	Elements should be of same data types.
Types-circular, priority, double ended queue.	Types-1D, 2D etc.	Only one type.	Types-single linked list, double linked list.

III. CONCLUSION

For a programmer one of the most basic and important aspects to decide is; on what data structure their program will be coded in, with the change of data structure the methodology and syntax of their program will change as well.

The key lies in choosing the right data structure for their projects purpose as this will enhance the efficiency of the computer program ease in understanding syntax and stability of the whole project.

A data structure plays a vital role in Big Data Handling, Cyber Security and IOT.

The processing speed component, data search and handling multiple requests from users in a network all depend on the type of data structure used and how it is used.

Specifically in a network of users the right data structure can lead to enhanced cyber security and ensure that the data stays within the network and is stored accurately.

All in all through this paper we can conclude that a data structure is a foundational requirement in cyber security and is the support system for various other aspects of today's modern technology.

IV. REFERENCES

[1]. Patel, Mayank. Data Structure and Algorithm With C. Educreation Publishing, 2018.

[2]. Bachman, Charles W. "Data structure diagrams." ACM SIGMIS Database: the DATABASE for Advances in Information Systems 1, no. 2 (1969): 4-10.

[3]. Kruse, Robert, and C. L. Tondo. Data structures and program design in

[4]. C. Pearson Education India, 2007.

[5]. Kruse, Robert, and C. L. Tondo. Data structures and program design in

[6]. C. Pearson Education India, 2007.

[7]. Sleator, Daniel D., and Robert EndreTarjan. "A data structure for dynamic trees." Journal of computer and system sciences 26, no. 3 (1983): 362- 391.

[8]. Hoel, Erik G., and Hanan Samet. "A qualitative comparison study of data structures for large line segment databases." In Proceedings of the 1992 ACM SIGMOD international conference on Management of data, pp. 205-214. 1992.

[9]. Forouzan, Behrouz A. Cryptography & network security. McGraw-Hill, Inc., 2007.

[10]. Stinson, Douglas Robert, and Maura Paterson. Cryptography: theory and practice. CRC press, 2018.

[11]. Wang, Xiao Shaun, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. "Oblivious data structures." In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 215-226. 2014.

[12]. Micciancio, Daniele. "Oblivious data structures: applications to cryptography." In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pp. 456-464. 1997.