International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 06 | June - 2023 SJIF Rating: 8.176 ISSN: 2582-3930

A Review on Practical Approach to Data Structures and Algorithms

Ms.Pooja[1], Ms.Sukhwinder Kaur[2], Ms.Harpreet Kaur[3] ,Ms.Mandeep Kaur[4]

[1]Ms.Pooja UCCA & Guru Kashi University
[2] Ms.Sukhwinder Kaur UCCA & Guru Kashi University
[3]Ms.Harpreet Kaur UCCA & Guru Kashi University
[4]Ms.Mandeep Kaur UCCA & Guru Kashi University

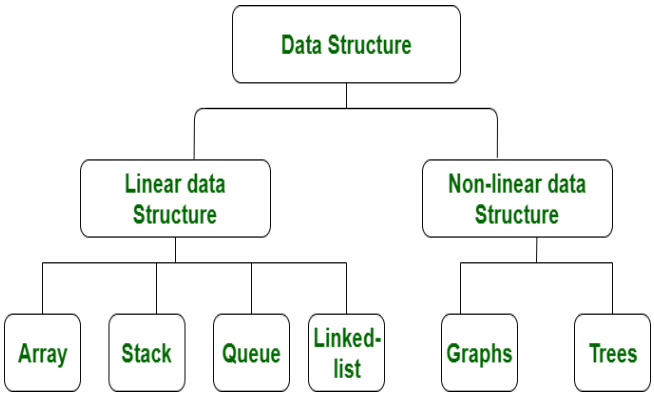
Abstract - Run in keeping with the significance. The first section of this paper defines the abecedarian terms utilized in this study's discussion of data structures. Other restrictions, similar memory operation, which will be pivotal, will affect in better handling times. The stylish data structures and styles, as opposed to hacking and killing a many statements with some smart coding. Abstract data types (ADT) are erected on top of data structures." The logical form of the data type is defined by the ADT. The physical shape of the data type is enforced by the data structure. Some data structures are extremely specialized to particular tasks, and different types of data structures are suitable for different feathers of operations. For case, B- tree indicators are constantly used by relational databases for data reclamation, whereas hash

***Key Words*:**

1. INTRODUCTION *(Size 11, Times New roman)*

A storehouse that's used to store and organize data is called a data structure. It's a system of setting up data on a computer to make it fluently accessible and over to date. A data structure is used for further than just data organization. Also, it's employed for data processing, reclamation, and archiving. Nearly all software systems and programmers that have been produced use several introductory and complex forms of data structures. Thus, we need to be well- clued in data structures.

Types of data structure& algorithms-



LINEAR DATA STRUCTURE

A linear data structure is one in which the data elements are ordered sequentially or linearly, with each element connected to its immediate neighbor. A single level is concerned in linear data structures. As a result of the linear organization of computer memory, linear data structures are simple to implement. Array, stack, queue, linked list, etc. are some examples of it.

1. Array

An array is a fundamental data structure used to store elements of the same type. It provides a way to organize and access data efficiently. Each element in an array is assigned a positive value called the index, which represents its position within the array. By using the index, we can easily locate and retrieve elements from the array. For example, if we need to store some data, an array can be a suitable choice.

2. Stack

The data structure known as a stack adheres to the principle of LIFO (Last In, First Out), where the most recently added element is the first one to be removed. Adding an element to a stack is called a "push" operation, while removing an element is called a "pop" operation. This concept can be illustrated by envisioning a stack of books. To access the book at the bottom of the stack, all the books above it must be carefully taken off first.

3. Queue

This structure is almost similar to the stack as the data is stored sequentially. The difference
is that the queue data structure follows FIFO which is the rule of First In-First Out where the first added element is to exit the queue first. Front and rear are the two terms to be used in a queue.
Enqueue is the insertion operation and dequeue is the deletion operation. The former is performed at the end of the queue and the latter is performed at the start end. The data structure might be explained with the example of people queuing up to ride a bus. The first person in the line will get the chance to exit the queue while the last person will be the last to exit.

4. Linked List

The structure that resembles a queue stores data in a sequential manner. However, unlike a stack, the queue follows the principle of FIFO (First In, First Out), where the first element added is the first one to be removed. In a queue, we use the terms "front" and "rear." The insertion operation is called "enqueue," which adds an element to the end of the queue, while the deletion operation is called "dequeue," which removes an element from the front end. To better understand this concept, imagine people lining up to board a bus. The person at the front of the line will be the first to board and exit the queue, while the person at the end of the line will be the last one to exit.

Non-linear data structure-

Non-linear data structures are data structures in which the data elements are not arranged sequentially or linearly. Unlike linear data structures, non-linear data structures do not involve a single level, which means that we cannot traverse all the elements in a single run. Implementing non-linear data structures can be more challenging compared to linear data structures. However, non-linear data structures make efficient use of computer memory when compared to linear data structures. Examples of non-linear data structures include trees and graphs.

1. Trees

A tree data structure is composed of interconnected nodes, forming a hierarchical relationship resembling that of a parent and child. The tree structure ensures that each parent-child node relationship is represented by a connection. There is a single unique path between the root node and any other node in the tree. Trees come in different types, such as AVL trees, binary trees, binary search trees, and more, each characterized by their specific structures and properties

2. Graphs

Graphs are a type of non-linear data structure that consists of a set of vertices and edges. The vertices, also known as nodes, are used to store data, while the edges represent the relationships between the vertices. Unlike trees, graphs do not have specific rules for connecting nodes, allowing for more flexible and diverse relationships. Graphs are commonly used to represent real-life problems such as social networks, telephone networks, and various other interconnected systems.

A practical approach

To approach data structures practically, follow these steps:

1. Understand the Fundamentals: Gain a clear understanding of basic data structures such as arrays, linked lists, stacks, queues, trees, and graphs. Learn their properties, operations, and time complexities.

2. Choose a Programming Language: Select a programming language suitable for data structure implementations, such as Python, Java, C++, or C#.

3. Implement Data Structures: Start implementing data structures based on your understanding. Begin with simpler structures like arrays and linked lists, and gradually move on to more complex ones. Write code for creating, manipulating, and accessing data within these structures.

4. Master Essential Operations: Ensure you can implement essential operations for each data structure. For example, focus on inserting, deleting, and accessing elements in an array, or traversal algorithms in trees.

5. Analyze Time and Space Complexities: Understand the time and space complexities of operations on data structures. This knowledge helps evaluate algorithm efficiency and choose the appropriate structure for specific scenarios.

6. Solve Problems and Practice: Apply your knowledge to solve programming problems and algorithmic challenges. Practice on websites like LeetCode, HackerRank, or Code Signal to reinforce your understanding and enhance problem-solving skills.

7. Explore Advanced Data Structures: Once comfortable with basics, explore advanced structures like heaps, hash tables, tries, AVL trees, or red-black trees. Understand their use cases, implementation details, and associated algorithms.

8. Consider Real-World Applications: Study how data structures are used in real-world applications. For example, understand how hash tables are utilized in databases or how graphs power social networks. This provides practical insights and performance optimization opportunities.

9. Learn from Existing Libraries: Study and utilize existing data structure libraries available in your chosen programming language. These libraries often offer efficient and optimized implementations, saving you time and effort.

10. Continuously Improve: Stay updated with the latest advancements in data structures and algorithms. Attend workshops, read books, and follow online resources to enhance your skills and stay current with industry

trends.

By following these steps, you can develop a practical approach to data structures and effectively utilize them to solve real-world problems.

Conclusion

This paper provided an introduction to the basics of data structures, but it's important to note that there is much more to explore in this field. We have laid a solid foundation to build upon. Data structures extend beyond just Stacks, Queues, and Linked Lists; it encompasses a vast area of study. Additional data structures include Maps, Hash Tables, Graphs, Trees, and more. Each data structure has its own advantages and disadvantages, and it should be selected based on the specific requirements of the application at hand.

A computer science student should possess knowledge of the fundamental data structures along with their associated operations. Many high-level and object-oriented programming languages, such as C#, Java, and Python, come equipped with built-in implementations of these data structures. However, it is still crucial to understand how these data structures work internally. Dynamic data structures necessitate dynamic storage allocation and reclamation. This can be done explicitly by the programmer or implicitly by the language itself.

Understanding the fundamentals of storage management is vital because these techniques greatly influence the behavior of programs. The underlying concept revolves around maintaining a pool of memory elements that can be utilized to store components of dynamic data structures as needed. Allocated memory can be returned to the pool when it is no longer required, allowing it to be reused efficiently.

References-

1.Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
2.Book of *"Data structures* "through C G. S Baluja
3.Algorithms, Part I" and "Algorithms, Part II" by Robert Sedgewick and Kevin WaynPieren Garry Department of computer science New York University.
4.Paul Xavier department of algorithms in c msterdam.
5.Surendra kumar Ahuja IIt delhi department of computer science delhi .
6.Nick jones department of data mining Australia.
7.Wikipedia sequential search