International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 04 | Apr 2021 www.irjet.net p-ISSN: 2395-0072

# Data Structures and its Applications in C

**Sthuti J[1], Namith C[2], Shanthanu Nagesh[3]**

*[1,2,3]Students, Department of Information Science and Engineering, Global Academy of Technology, Bengaluru.*

**Abstract –** *In this modern world of technology, data must be used judiciously for which we need the most efficient ways of storing and organizing data so that we can reduce time complexities and space complexities. This can be achieved by choosing the most suitable data structure for our problem. This paper is mainly focused on the types of data structures, their organization, operations and applications.*

***Key words:*** **array, nodes, FIFO, LIFO, front, top, rear, link**

## 1. Introduction

Data structures are pre-determined ways of arranging memory locations into which logically related data is insertedfor easy handling. They are classified into primitive and non-primitive data structures. Primitive data structures are basic data structures which cannot be broken down into simpler datatypes whereas non-primitive datatypes can be further simplified as they are built using primitive datatypes.

## 2. Literature Survey

Data structures are one among the most fundamental concepts for a programmer. The implementation phase of any project would start off by the decision of data structure to be used. It gives an effective way for memory management and helps in analyzing the real-world problems in a much simpler way.

In [1], the author speaks about the analysis of array operations on the basis of canonical abstractions and numeric domains. The author speaks about the scalar variables and how the array can be partitioned into different elements with a particular index. In [2], Lester Leong gives a brief idea about the array and all the basic operations that can be carried out on arrays.

The author tells us about the best way of choosing the data structure for a particular problem is based on the number of steps the operations.

In [7], the author has compared stacks to a set of disks in a CD pack and queue to a real-world queue in which a new person can join only at the
rear end.

According to [10], a linked list is considered to be collection of linearly arranged elements where the data elements are called as nodes. In [6] and [7], queues and stacks are compared to each other and their applications are explained. In [8], the author throws some light on the properties of queues and their operations. It also lists some real-world applications of queues.

The basic terminologies used in trees are clearly described in [11] and [12]. In [11], Reema Thareja explains all the basic operations on trees. It also shows how trees are represented in the form of arrays and linked lists. They also tell us about the important real-world applications of trees.

In [13],Estefania Cassiagena Navone gives a basic idea about how the graphs are visualized in this real world and the basic purpose of using this graph. She also explains the elementary types of graphs and some basic terminologies related to it. The author states that the graphs are fundamentally used to set up the connections between the different elements and also talks about how to find and analyze these elements.

## 3. Arrays

Array is a combination of **similar data items** which are collected together. Each element is referred to by their **index or subscript**. The index always starts with the value 0 and ends with n-1.
For example, consider an array of 4 employees in an organization

| | | | |
|---|---|---|---|
| 4567 | 4568 | 4672 | 4831 |

Fig 3.0: Array representation for 4 employees

Arrays can be classified into: (i) One-Dimensional Array and (ii) Multi-Dimensional Array

### 3.1 Single-Dimensional Array

It is a linear and contiguous list consisting of related and similar data types.
Consider the image shown below

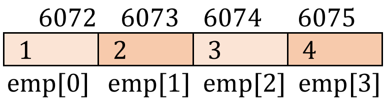


Fig 3.1 : 1-D array representation with address

The technical way of array definition includes a variable type and a name with a size which tells how many data items the array will contain.

***type array_name*[size];**

### 3.2 Multi-Dimensional Array

A Multi-dimensional array is a type of array which has two or more dimension to define itself. A Multi-Dimensional array can be declared with the number of rows and he number of columns it contains. The syntax for Multi-Dimensional array declaration: ***data_type array_name [no. of rows][no. of columns];*** To find the number of elements present in the array, we need to multiply the number of row =s with the number of columns. The general form of representing a 2D array is:

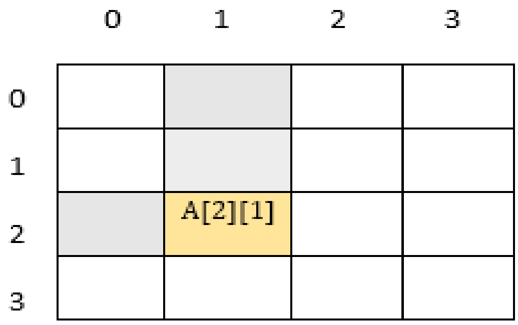


Fig 3.2

Representation of 2D array

### 3.3 Operations on Array

Array operations are the fundamental changes and updation done on the array. The fundamental operations that can be done on arrays include the following:

- Insertion
- Deletion
- Display
- Sorting
- Searching

#### 3.3.1 Insertion

When an array consists of N elements, if there is a requirement for adding a few more elements to the array for better and efficient handling, a new element can be added to the array.
When insertion of a new element to an array is necessary, the size of the array will be increased. The condition for insertion of an element is that the array should not be full.

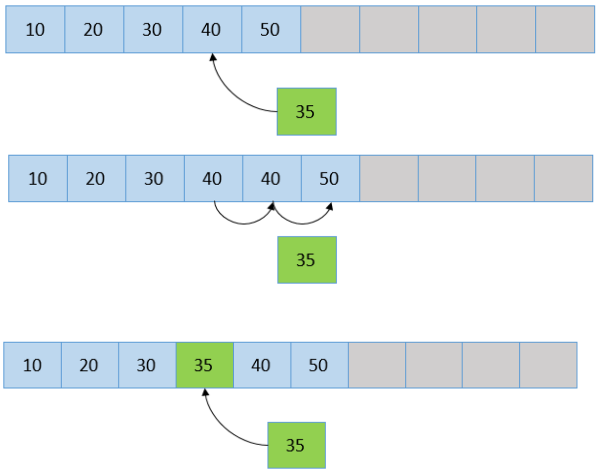


Fig 3.3 Inserting an element into the array.

Here, 35 is the new element to be inserted into the array at position 4. All the elements are shifted one place starting 4 till the size of the array(5 here). The element that is to be inserted(35 at position 4) is inserted and the size of the entire array is increased by 1.

#### 3.3.2 Deletion

Deletion is the process of removing a value from an array. It basically is like insertion in an opposite kind of a way. Deletion at a specified position deletes a particular element from the position and then covers the gap by shifting the elements one position backwards.

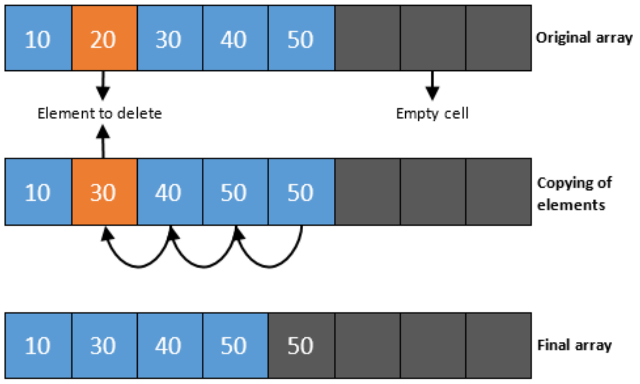


Fig 3.4: Representation of deletion in an array.

Here the element to be deleted from the array is 20 at position 2. Starting from 2, all the elements till size of array(5 here) are shifted one position to the left to cover the gap of the deleted element.The effective size of the array is then reduced by 1.

### 3.3.3 Display

Display is the process of traversing through the array and printing all the elements till the end. The display operation can be performed on an array only when there is at least 1 element in an array. The counter i iterates through all the elements in the array till it reaches the end of the array and prints all the elements one by one.

### 3.3.4 Sorting

Sorting is a process where in an important task is performed that sorts the elements of an array in a certain order. Sorting of data in an array can be performed in various ways. For example, elements in an array can be arranged in low to high fashion or from high to low fashion or based on any other particular condition. Sorting involves two main steps which is comparing two items and swap these two items on basis of the condition specified. It keeps repeating until all the elements in an array is sorted and the condition is satisfied.

- **Bubble Sort**

  In Bubble sort, it compares the first two elements of the array and if the first is greater than the second, it interchanges them. It then proceeds for every pair of adjacent elements till the end of the array. It again starts with the first two elements in the array, until all the elements are sorted in the array.

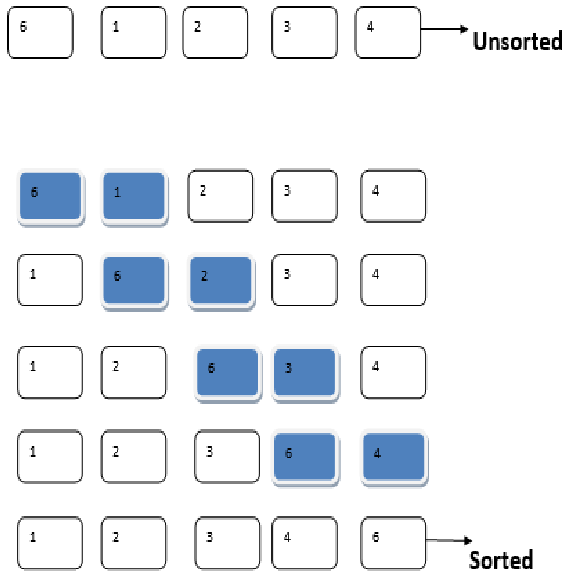


Fig 3.5 : Bubble sort

The number of passes for this type of sorting is n-1.

### Insertion sort

It works on the principle by taking elements from the array one by one and then placing them in their correct position into the new sorted array.
To summarize, an insertion sort requires (n-1) passes where n is the number of elements in an array.

### 3.3.5 Searching

Searching is the process in which an element is searched in an array. Searching can effectively be performed in 2 ways: (1) Linear search (2) Binary Search

- **Linear Search**

  Linear search is a simple and quite quick searching algorithm. It is a linear search which is performed on array of numbers that are sorted or unsorted. It checks each and every number of the array to search for a specified number. If the number is found, then that number is printed and tells the user that it is found. For an array with n numbers, the easiest iteration is when the number to be searched is in the first position, here only one iteration is needed. If the value is not present in the list, it takes n comparisons which is going to take time.

**Step 1**: initialize variable i to1.
**Step 2**: check if i greater than n which is the size of array. If it is then, then go to step 7
**Step 3**: if array[i] and the value of x are equal then, go to step 6
**Step 4**: increment i to i+1
**Step 5**: Go back to step 2.
**Step 6**: If the element is found, print it and go to step 8
**Step 7**: If not found, Print element is not found
**Step 8**: Exit

- **Binary search**

Binary search is one of the searching algorithms where in, the array is divided into 2 sets with a mid-number for the array. The key to be searched is first compared to the mid element. If the element is present in the mid position, print it is found. If the element is not found in the middle, it is the checked if the element is present in the lower half of the array or the other half of the array and the same process is repeated until the element is found.
**Step 1 –** Take the input from the user the element to be searched.
**Step 2 –** Identify the middle element is in the array.
**Step 3 –** Compare the search element with the middle element in the array.
**Step 4 –** If the element is the middle element print Found and exit.
**Step 5 –** If the key is not matching, then check if the key is in the lower half of the array or in the upper half.
**Step 6 –** If the search is successful in the lower half of the array, repeat the steps 2,3,4 and 5 for the lower half of the array.
**Step 7 –** If the element to be searched is larger than the middle element, repeat steps 2,3,4 and 5 for the higher half of the array.
**Step 8 –** Repeat the process unitl the element is found.
**Step 9 –** If the element is not found, print the element is not there in array.

### 3.4 Application of Arrays

- Array can be used to store similar data types in a sequential order.
- Can be used while sorting elements.
- All Matrix operations can be performed on arrays.

## 4. Stacks

A stack is an arrangement of elements which can be represented using arrays where both insertion and deletion takes place at the same end. This is the **L**ast **I**n First **O**ut (LIFO) principle. The stack follows LIFO principle. According to [6] In stack all insertions and deletion of data items are done at one end called Top.

Stack operations are operations that are performed on stacks. These operations are push and pop.

Inserting an element into stack is called **push** operation. Only one element can be inserted at a time and it has to be inserted only from top of the stack. When the value of variable top is equal to the size of the array, further insertion cannot take place. . Further trying to insert an element to a stack results in an **overflow** of stack.

Stack contains after inserting 3 elements 7, 8 and 9.

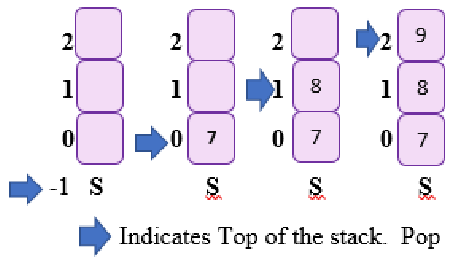


Fig 4.0: Pushing operation in a stack

Deleting an element from stack is called **pop** operation. Only one element can be deleted at a time and it has to be deleted from the top of the stack. Once the stack is empty, it is not possible to delete any element. Further trying to delete an element from the stack, results in an **underflow**.

Performing pop operation when stack already contains 7, 8 and 9 is shown below.

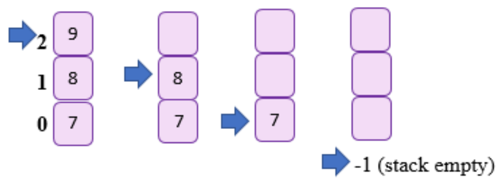


Fig 4.1: Popping operation in a stack

### 4.1 Application of stack

When mathematical expression in program is written in the program, we use infix expression. Using **stacks** these expressions will be converted into equivalent machine instructions by the compiler. We can efficiently convert the expression from **infix** to **postfix, infix** to **prefix, postfix** to **infix, postfix** to **prefix, postfix** to **infix** and **postfix** to **prefix** using stacks.

Stack can be used to evaluate expressions. It can mainly be used to evaluate postfix expression and prefix expressions. **Undo** mechanism in text editors, **Backtracking** and **Language processing** are some applications of stacks.

### 4.2 IMPLEMENTATION OF STACKS

#### 4.2.1 Array-based implementation

In this we maintain the following fields: according to [2] an array A of a default size (>=1), the variable **top** that refers to top element in the stack and capacity that refers to array **size**. The value of variable top changes from -1 to (size of array-1) . We say that stack is empty when top=-1, and the stack is full when top=capacity-1. Hence, this a **static** stack implementation.

#### 4.2.2 Linked list-based implementation

In this implementation a stack contains a top pointer which is called **head** of the stack. Push and pop operations takes place at the head of the list. Advantage of using linked list is that it is possible to make an efficient use of the memory. Hence, this is a **dynamic** stack implementation.

## 5. Queues

Queue is a linear data structure which follows **F**irst **I**n **F**irst **O**ut principle (FIFO) which means that the first element which will be inserted into the queue will be the first element to be deleted. According to [8], a queue is an ordered collection of items where the addition of new items happens at one end and removal happens from the other end. To make the operations on queues easier, we keep track of the first element of the queue as 'Front' and the last element as 'Rear'.

### 5.1 Operations on Queues

Enqueue: Adding an element into the queue is known as enqueue.
Dequeue: Deleting an element from the queue is known as dequeue. To make enqueuing and dequeuing simple, we use some additional operations like isFull and isEmpty to check if the queue is empty or full.

### 5.2 Types of Queues

#### 5.2.1 Simple queue

In simple queues, the enqueue operation is carried out at the front end of the queue and dequeue operation is carried out at the rear end. It strictly follows the first in first out principle.

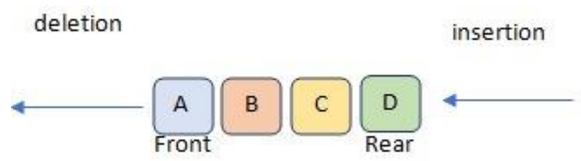


Fig 5.0 – Representation of a simple queue

#### 5.2.2 Double ended queue

In double ended queues, enqueue and dequeue operations can be carried out on both the ends of the queue. It is more flexible since it can be modified easily.

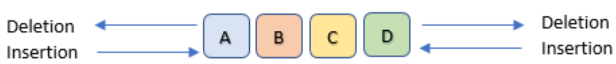


Fig 5.1: Double ended queue.

#### 5.2.3 Priority queues

In priority queues, each element will be assigned with a priority number and the element with the highest priority number will be the first element to be dequeued. If two elements in the queue have the same priority, then those elements will be dequeued according to their order in the queue.

#### 5.2.4 Circular queues

Theoretically, even if the queue has space available, it is not be possible to enqueue once the end of the memory allocated for the queue is reached. To avoid this situation, we use circular queues which enable us to use the space of the previously dequeued elements.

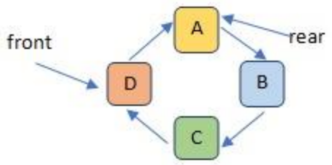


Fig 5.2: representation of a circular queue

### 5.3 Applications

According to [8] and [9], Queues can be used for implementing Breadth First Search traversal in graphs, handling interrupts in real-time systems, call-holding in call center systems till a service representative is free. FIFO principle of queues can be used in CPU scheduling of instructions on first come first serve basis. It can be used in automated appointment administration systems to mimic the real-world queues. Other applications of queues are handling interrupts, print spooling, transmitting packets of information over a computer network.

## 6. Linked List

A Linked List is sequential collection of data items where the order is not determined by the arrangement in the physical memory. Instead, each element points to next node. It is represented as a collection of nodes where one node points to its next node.

Each node contains two parts, one represents the data and the other contains the reference to the next node. This structure is an efficient way of representing a linked list.

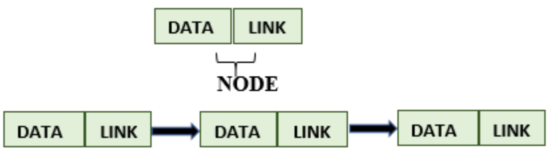


Fig 6.0: Linked list representation.

Linked List is classified as **Singly** linked list, **Doubly** linked list, **Circular** linked list and **Header** linked list.

### 6.1 TYPES OF LINKED LISTS

#### 6.1.0 Singly linked list

A singly linked list is a linked list, where each node has a designated field called **link,** which contains address of the next node. It may have a collection of nodes with various field. But each node should have only one link.

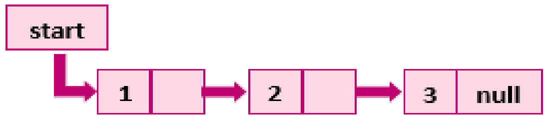


Fig 6.0: Singly linear linked list representation.

#### 6.1.2 Doubly linked list

A doubly linear linked list contains two pointers and a data part where one of the pointer points to the preceding node and the other pointer will point to the subsequent node.

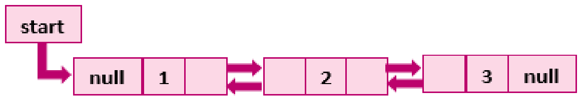


Fig 6.1: Doubly linear linked list representation.

#### 6.1.3 Circular linked list

In a circular linked list, the address part of the last node contains the address of the first node. Here, the traverse of the list is possible in any of the directions, forward or backwards. Circular linked has no beginning and no ending.

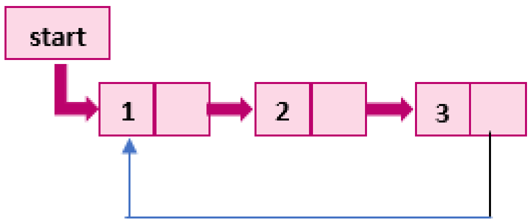


Fig6.2: Representation of circular linked list

#### 6.1.4 Circular doubly linked list

A circular double linked list is more complicated form of linked list, that has a pointer to next and previous node in the list. The distinction between a double linked list and a circular double linked list, according to [11], is that the circular double linked list does not include **NULL** in the previous and next fields of the first node, the reference of first node is found in the last node.

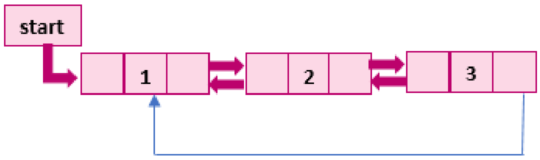


Fig 6.3: Representation of a doubly circular linked list

### 6.1.5 Header linked list

It is a special type of linked list which contains a header node at the beginning of the list. Start of header linked list will point to the first node of the list but start contains the address of the header node.

**Ground header node** it stores NULL in the last node of the linked list.

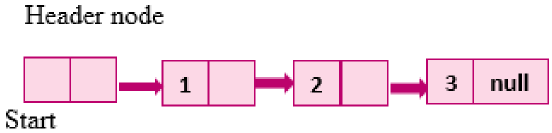


Fig 6.4:Representation of linked list with header node.

### 6.1.6 Circular header linked list

In header linked list, the header node's address is stored in the last node of the next link.

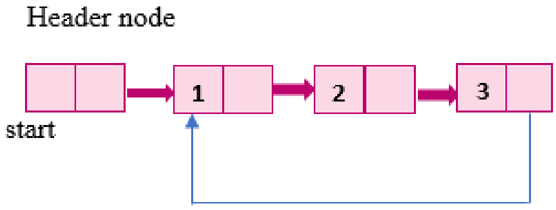


Fig 6.5: Representation linked list with circular header node.

### 6.2 Application of linked list

Linked list is used in the Implementation of **stacks**, **queues** and **graphs**. It is also used in performing **arithmetic** operation and in the Representing **sparse** matrices. Sparse matrices are a matrix that contains most elements as zero and very few non zero elements.

Most of the time linked lists are used in Music players. Whenever a song is played on a music player, the song has a link to the previous song as well as the next song. Linked lists are also used in Image viewer. Previous and next image are linked, can be accessed by next and previous button.

## 7. Trees

According to [11] and [12], a tree can be defined as a recursive set of one or more nodes in which the top node is considered to be the root of the tree and the rest of the nodes are non-empty sets of which is a sub-tree of a root. Each node is associated with a value and list of references to its children nodes.

### 7.1 Basic terminology

**Root node**: It is the single node which is present at the top of tree which is NULL when the tree is empty. In fig 7.0, A is the root node.
**Leaf node**: It is a node which has no children nodes. In fig 8.0, C, E and D are children nodes.
**Sub tree**: When the root node is not NULL, the trees obtained by considering the children of the root node as root nodes will become the sub-trees. In fig 8.0, {B, C, E}, {D} are the sub trees.
Ancestor node: It is a node which is accessible by proceeding repeatedly from child node to parent node. In fig 8.0, A is an ancestor node of E.
**Descendant node**: It is a node which is accessible by proceeding repeatedly from parent to child. In fig 8.0,
E is a descendant node of A.
**Path**: It is a sequence of consecutive edges. In fig 8.0, AB-BC is a path from A to C.
**Depth**: Depth of a node is the minimum number of edges present between the root node and that particular node. In fig 8.0, the depth of C is 2.
**Height of a tree**: It is the number of edges present between the root node and the deepest node in the tree. In fig 7.0, height of the tree is 2.

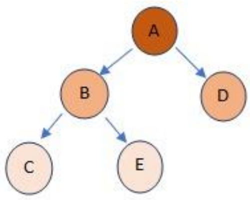


Fig 7.0 : Representation of a tree data structure

### 7.2 Important types of trees

**General tree**: It is a type of tree which can have zero or more child nodes. There is no limit for the number of child nodes per node.
**Binary tree**: It is a type of tree which has at most two children nodes per node. Each node will have a value and pointers pointing to its right child and left child.
**Binary search tree**: It is a type of binary tree in which the left sub tree contains only the nodes with values lesser than the current node (node being traversed) and the right sub-tree contains nodes with values greater than the current node.
Traversal of binary trees

| Type | Sequence | Traversal of Fig 5.1 |
|---|---|---|
| Inorder | Left node->Root->Right node | C-B-E-A-D |
| Preorder | Root->Left node->Right node | A-B-C-E-D |
| Postorder | Left node->Right node->Root | C-E-B-D-A |

Fig 7.1: Types of binary trees traversals.

### 7.3 Representation of Binary Trees

#### 7.3.1 Array representation

Binary trees are stored in a one-dimensional array with the root node as the first element and the nodes being arranged in a left to right order into their corresponding array indices. The tree in fig 8.0 is represented as in fig 8.2.

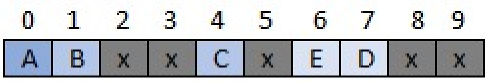


Fig 7.2: Representation of array using Binary Tree

#### 7.3.2 Linked list representation

Binary trees can also be represented using doubly linked lists. Each node will contain a value and the address of its children nodes. The leaf nodes will have NULL in the address section.
The linked list representation of fig 8.0 is shown in fig 8.3.

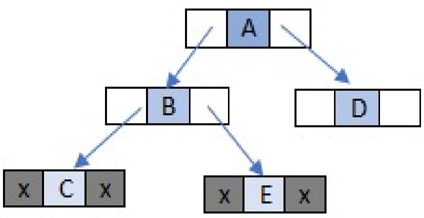


Fig 7.3 : Linked list representation of binary trees

### 7.4 Application of trees

As in [11] and [12], Trees can be used in compiler construction. Binary searched trees can be used to store data in a way which is easily reachable for fast insertion, deletion and searching. Trees can be used to represent hierarchical data like file system directories.

## 8. Graphs

**Graph** is a non-linear fundamental and abstract **data structure** which is made up of nodes and edges.
It can technically be defined as G=(V,E) where V<E.
Where V is defined as the Vertex of the graph and E is defined as the Edge of the graph.

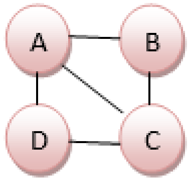


Fig 8.0: Representation of graph data structure

Here the points A,B,C and D are called the **vertices** and the lines connecting these vertices are called the **edges.**

### 8.1 Basic terminology

**Vertex**: A vertex is a basically called a node. In a graph, it is normally represented by a circle and a label.
Example:


Fig 8.1: Representation of vertices

Here 2 nodes are identified as V={N,O}.
**Edge**: If there are 2 vertices present in a graph, then an arc or a line joining these 2 vertices is called an edge.

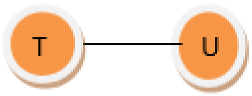


Fig 8.2: Representation of Edges

Here the two vertices are T and U and the line connecting these 2 lines is called an Edge.

**Directed graph**: A graph in which every edge has a direction towards or away from the Vertex is called a directed graph.

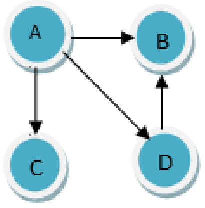


Fig 8.3: Representation of directed graphs

**Undirected graph**: A graph that is defined by G=(V,E) in which every edge is undirected or just a connection between 2 vertices is called an undirected graph.

**Weighted graph**: Let a graph be defined in such a way that G=(V,E), in which a number is assigned to each edge. These numbers are called costs or weights in a graph.

**Degree of a node**: The number of children that a node has is formally defined as Degree of a node.

Degree can be defined in two ways:

**Indegree**: It can be defined as the total number of edges that are incoming from another node to one particular node is called Indegree

**Outdegree**: The total number of edges that are going from a particular node to another node is called the Outdegree.

## 8.2 Graph Representation

The two most common ways of representing a graph is using:

(a) Adjacency Matrix

(b) Adjacency Lists

### 8.2.1 Adjacency Matrix representation

The Matrix can be used to pictorially represent the interconnection of nodes and vertices in a graph. By the definition of a graph, two nodes are said to be connected in a graph if there is an edge between the two nodes. If a node A is adjacent to B, we can get from one node to another by traversing the edge between them. The matrix input will have the value **1** if two nodes are adjacent to each other. If the nodes are not adjacent, the matrix input will have the value **0.**

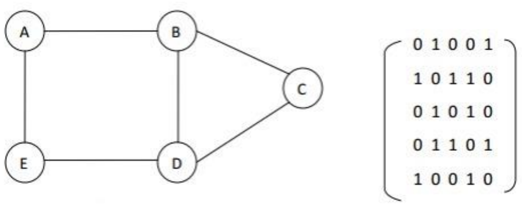


Fig 8.4: Adjacency matrix representation

### 8.2.2 Adjacency List representation

A linked list can be used to represent the connections in a graph. Here, every node is linked to its adjacent node. Linked list can easily be used to pictorially represent the adjacent nodes in a graph which makes it more efficient.

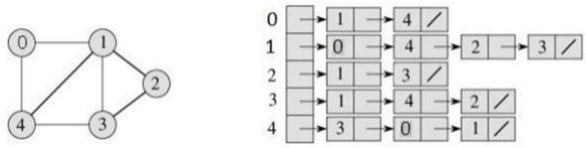

Fig 8.5: Representation of Adjacency list.

## 8.3 Graph traversals algorithms

Graph traversing is the process of visiting each vertex of a graph

### 8.3.1 Breadth-first search (BFS)

Breadth-first search (BFS) is one of the search algorithm in graph that begins at a given node called the root node which is specified by the user and finds all of its all neighboring nodes. For all the nodes, the unvisited node is found out. Here, we begin with the node A and then all the neighbors of this node are checked. After this, we search for the neighbors of neighbors of A and so on. This is can be done with the help of a queue.

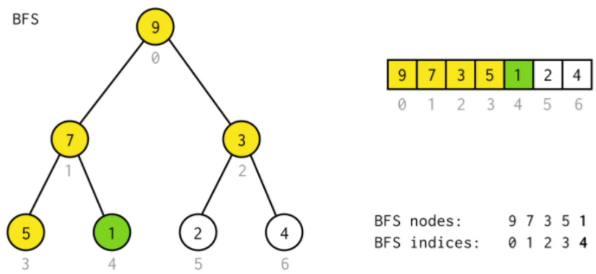


Fig 8.6 : Representation of BFS algorithm

### 8.3.2 Depth-first search (DFS)

The search starts by a node that has been specified by the user which the finds all the possible nodes along each branch of the graph before backtracking. The search starts by checking all the marked nodes and find continues until it finds no adjacent node for that particular node. The final step is to then backtrack and check for unmarked nodes It then notes the number of nodes in that traversal and prints those nodes.

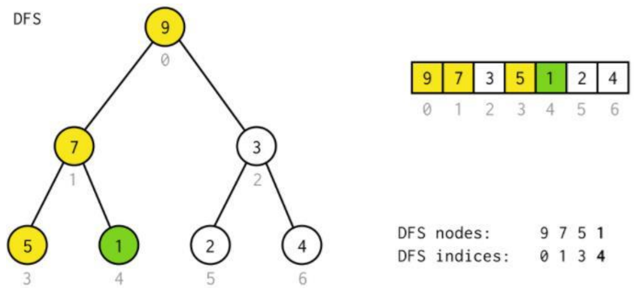

Fig 8.7: Representation of DFS algorithm

### 9.4 Applications of Graph

- The graphs are effectively used to calculate the shortest path from the departure point to the arrival point which makes it time effective. This is the fundamental use of **GPS**.
- The way different systems are connected and how **network** is connected between different deices is pictorially represented using Graphs.
- To find the best result for the user, **Google** uses Graph to find it.

## 9. Conclusion

In this paper, we have explained the different types of data structures and their operations. It also clearly depicts the implementation of each data structure with the help of diagrams. It also explains how each data structure is applied in the real world. Data structures are the best arrows to have in our quiver of computer science. They have their own advantages and disadvantages. Fortunately, the number of advantages exceed the number of disadvantages. It is important to choose the correct type of data structure for our problem statement.

## 10. References

[1] Denis Gopan, Thomas Reps and MoolySagiv: "Numeric Analysis of Array Operations" https://research.cs.wisc.edu/wpis/papers/tr1516.pdf

[2] Lester Leong: "A Guide to Arrays andOperations (Data Structures)" https://towardsdatascience.com/a-guide-to-arrays-and-operations-data-structures-f0671028ed71

[3] Monika Yadav: "ARRAYS: REVIEW" http://ijirt.org/master/publishedpaper/IJIRT142671_PAPER.pdf

[4] Ahmad Shoaib Zia: "A Survey on Different Searching Algorithms" https://www.irjet.net/archives/V7/i1/IRJET-V7I1275.pdf

[5] Rahul Mehra. "Research Paper on Sorting Algorithms JariyaPhongsai". https://www.cse.iitk.ac.in/users/cs300/2014/home/~rahume/cs300A/techpaper-review/5A.pdf

[6] Nitesh, Manbir Singh, Rahul Yadav. "Research paper on stack and queue". http://www.ijirt.org/master/publishedpaper/IJIRT101357_PAPER.pdf

[7] Regi Maliyekkal S, Anila M. "Comparison of Stack and queue data structures" https://www.irjet.net/archives/V6/i6/IRJET-V6I6737.pdf

[8] Anoop, Sunil Rai. "A Comparative study of stack and queues in data structure" http://ijirt.org/master/publishedpaper/IJIRT101022_PAPER.pdf

[9] A. Jain, U. Kumar. "research paper on queues" http://ijirt.org/master/publishedpaper/IJIRT100828_PAPER.pdf

[10] Dr.Mahesh K Kaluti, Govindraju Y, Shashank A R, Harshith K S. "Dynamic Implementation of stacks using single linked list" https://www.irjet.net/archives/V5/i3/IRJET-V5I3434.pdf

[11] "Data structures using C second edition -Reema Thareja".

[12] Rubi Dhankar, Sapna Kamra, Vishal Jangra. "Tree concept in data structure" http://ijirt.org/master/publishedpaper/IJIRT101212_PAPER.pdf

[13]Estefania CassiagenaNavone: "DataStructures 101: Graphs—A Visual Introduction for Beginners"

https://www.freecodecamp.org/news/data-structures-101-graphs-a-visual-introduction-for-beginners-6d88f36ec768/