

Data Structures and Algorithms

What is Data Structure?

It is a way to store and organize the data, so that it can be used efficiently.

Examples

- ① Array ③ Structure ⑤ Stack ⑦ Graph
- ② pointer ④ LinkedList ⑥ Queue ⑧ Searching
- ⑨ Sorting

* DS is a set of algorithms that we can use in any PL to structure data in memory.

What is abstract datatype?

* To keep the data structured in memory, abstract data type concept is been introduced, the ADT is bound by set of rules.

Data Structure

Primitive Data Structure

- int
- char
- float
- double
- pointer

(can hold a single value)

Non Primitive Data Structure

- [Linear DS]
 - Arrays
 - LinkedList
 - Stacks
 - Queues

(store data in seq)
Linear

Non Linear DS

- trees
- graphs

(element connected with n number of elements)

Data Structures

- ↳ Static DS (size allocated at compile time)
Maximum size is Fixed
- ↳ Dynamic DS (size allocated at runtime)
Maximum size is Flexible

Operations on DS

- ① Searching
- ② Sorting
- ③ Insertion
- ④ updation
- ⑤ Deletion

Necessity of DS

- * Store the data efficiently in terms of time and space
- * We require some DS to implement another DS a particular ADT
- * Ex: Stack (ADT) created / implemented using LL (DS)

ADT \Rightarrow Blueprint

DS \Rightarrow Implementation

- * Selection of DS to implement ADT depends on user requirement (Time / space)

Advantages of DS

- ① Efficiency (Efficient DS \rightarrow Efficient time & spaces)
- ② Reusability
 - ↳ Create an Interface by using Efficient DS, provide that to client, He uses everytime
- ③ Abstraction
 - ↳ Implement stack using array / LL, provide the interface to user, user don't have implementation details hence Abstraction

Main AIM: Store and retrieve as fast as possible

Basic Terminology

- ① Data: Elementary value or collection of values
- ② Data Item: Single unit of value
- ③ Group Items: Data items that have subordinate D.Items
Ex: Employee Name (First, Last, Middle Name)
- ④ Elementary Item: DataItem which are unable to Divide (EID)
- ⑤ Entity: Object that has a distinct identity (Person, Places)
- ⑥ Attribute: Characteristic of an Entity

Datastructures \Rightarrow Allows us store & organise data

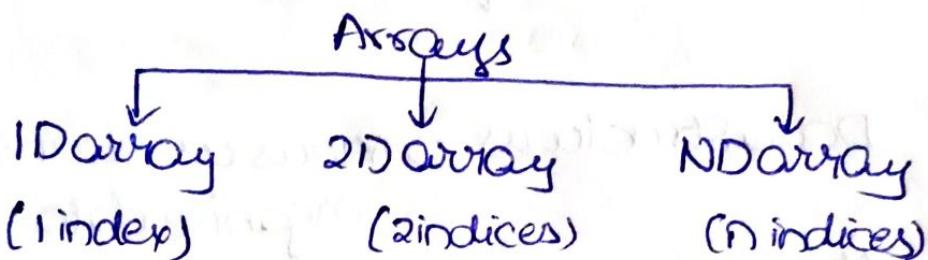
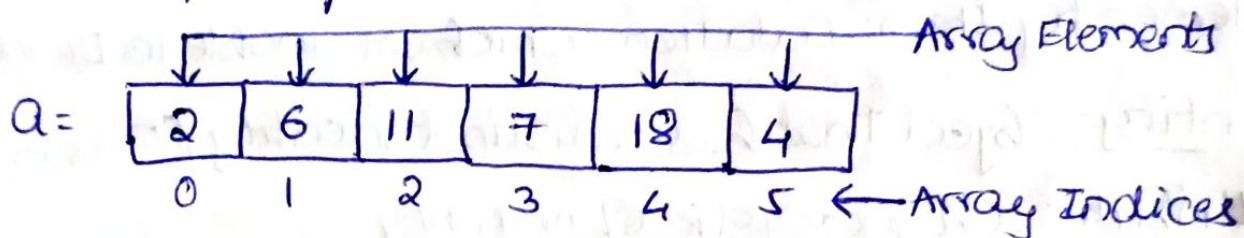
Algorithms \Rightarrow process the data meaningfully

- * Non Primitive Data Structure are data structures derived from Primitive DS
- * NPDS forms set of data elements that is either group of homogeneous/heterogeneous Data structure
 - ↳ same DT form as grp \hookrightarrow diff DT forms as grp

Linear Data Structures

① Array

- * Collect Multiple data elements of the same datatype into one variable.
- * Data is stored in contiguous memory location, so retrieving randomly through index based on array variable is possible.



Applications

- ① Store list of data elements of same type
- ② use as auxiliary storage for other data structures
- ③ store data elements of Binary tree of Fixed count

② Linked List

- * Singly linked list:
- * Doubly linked list
- * Circular linked list

Applications of Linked List:

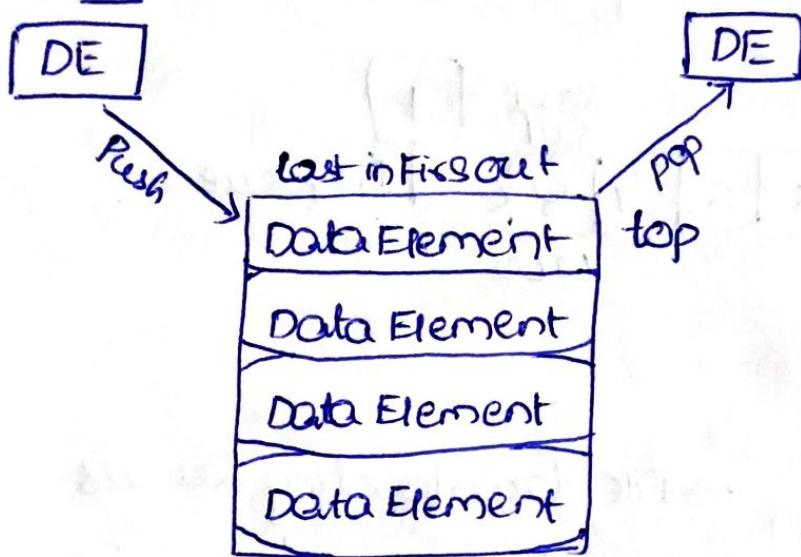
- * Helps us implement stacks, queues, binary trees and graphs of predefined size.
- * OS functionality, Dynamic Mem Managt
- * Slideshow functionality of PPT
- * First slide → last slide
- * DLL → Browser Front & Back navigation

Stack

- * Linear Data Structure that follows LIFO
- * Insertion & Deletion only from top end.
- * Implemented using
 - ↳ contiguous memory (Array)
 - ↳ Non contiguous Memory (linked list)

* Can access only Stack's top at any time

operations:



Applications:

- * Temporary storage for recursive operations, function calls, nested operations
- * Evaluate arithmetic Expressions
- * Infix Exp \rightarrow Postfix Exp
- * Match parenthesis
- * Reverse a string
- * Backtracking
- * DFS in graph & tree traversal
- * undo/redo func

Queues

- * Linear Data Structure
- * Insertion done at one end
- * Deletion done at opposite end
- * First in First Out

Can be implemented by

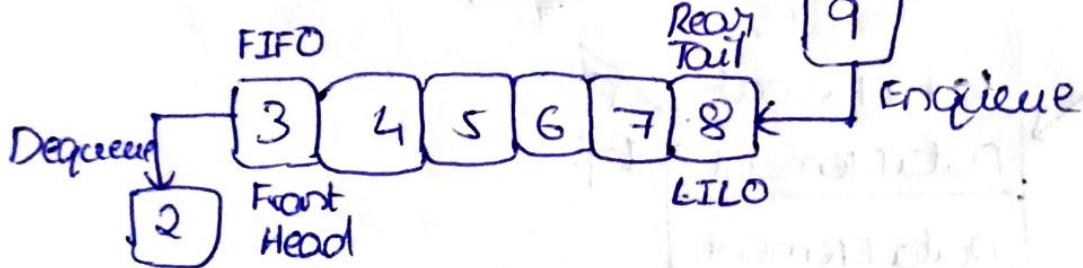
- Arrays
- Linked Lists
- Stacks

Real life Examples

ticket counter

Escalator

car wash



Applications

→ BFS in Graphs

→ File download Queues

→ Job scheduling operations

→ Handling interrupts

→ CPU/Job/Disk Scheduling

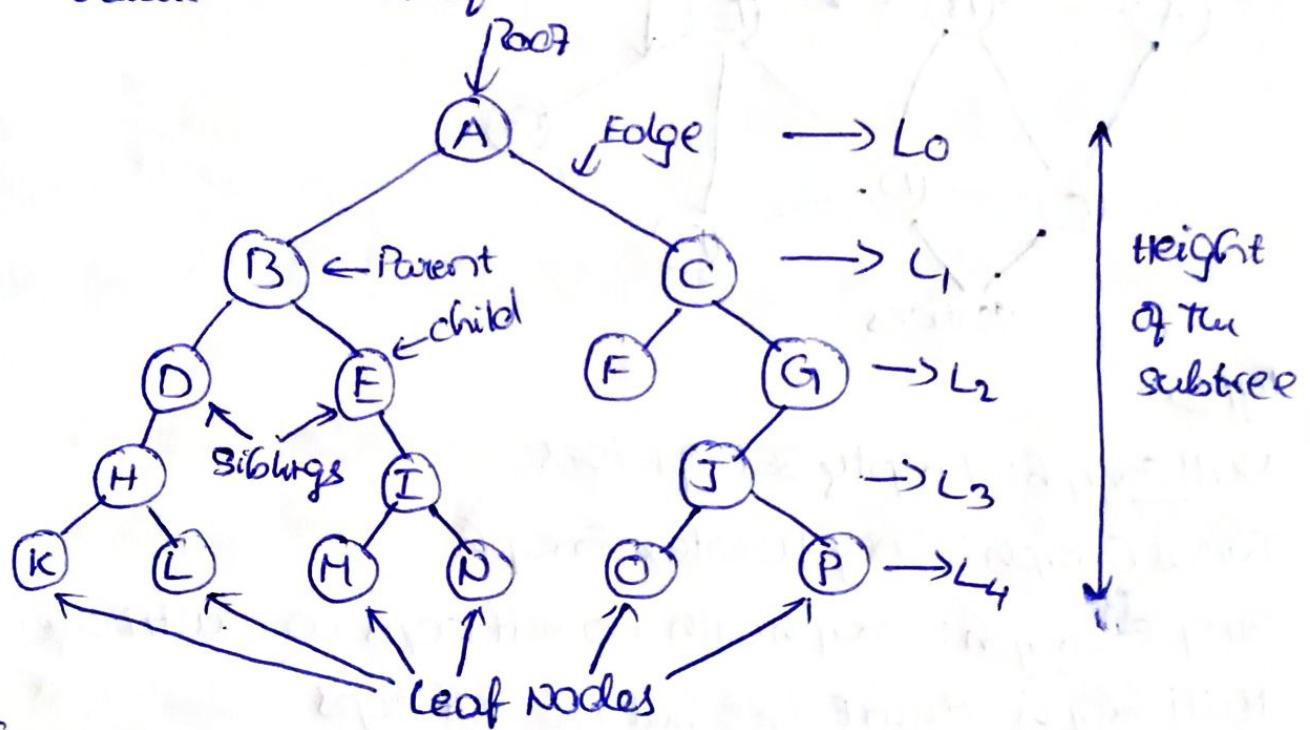
generated by user apps

Non Linear Data Structure

- * Data elements not arranged in sequential order.
- * Insertion & Removal is not that easy, hierarchical dependency between data items

Trees

- * collection of Nodes such that each node of tree stores a value and list of references to other nodes (children)



Types of Tree:

Binary Tree: 1 Parent node \rightarrow atmost 2 children

Binary Search Tree: can maintain sorted list of numbers

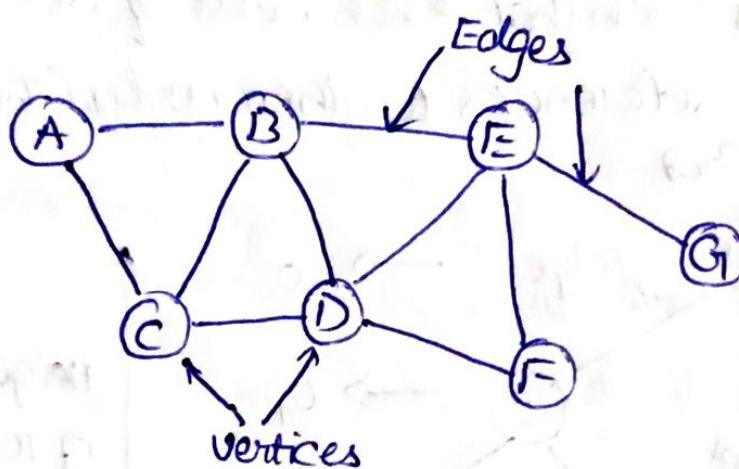
AVL Tree: Self Balancing Binary Search tree, Each node have Balance Factor (-1, 0, 1)

BTree: Similar to AVL Tree, Each node can have more than two children.

Graphs

* Finite nodes or vertices and the edges connecting them.

$G = (V, E) \Rightarrow$ set of vertices & Edges



Types

Null Graph: Empty set of edges

Trivial Graph: Only 1 vertex Graph

Simple Graph: Graph with no self loops no multi edges

Multi Graph: Multi edges but no self loops

Pseudo Graph: self loops & multi edges

Non-Directed Graph: non directed edges

Directed Graph: directed edges

Connected Graph: atleast a single path b/w every pair of vertices

Disconnected Graph: atleast one pair of vertices doesn't have edge

Regular Graph: all vertices have same degree

Complete Graph: all vertices have an edge b/w every pair of vertices

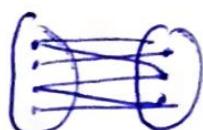
Cycle graph: At least 3 vertices and edges form a cycle

Cyclic graph: At least one cycle exists

Acyclic graph: zero cycles

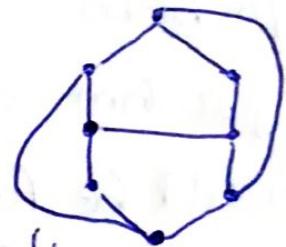
Finite / Infinite Graph: Finite / Infinite number of vertices / edges

Bipartite Graph: vertices can be divided into 2 sets
Set A vertices can be connected to Set B vertices



Planar graph: If we can "draw in a single plane" with
"two edges intersecting to Each Other"

Euler graph: All vertices are even degrees



Two Edges intersect
to Each Other

- ① Traversal
- ② Search
- ③ Insertion
- ④ Deletion
- ⑤ Sorting
- ⑥ Merge
- ⑦ Selection
- ⑧ update
- ⑨ splitting

Important points:

- * All Data Structures are Examples of ADT
- * If user want to store the data into memory, we provide array or LL, user don't know the implementation taken or this is main idea of Abstract Data type

Application of DS

- Rep info of DB
- search through org data
- Generate new data
- Encrypt & Decrypt data

Algorithm

- * set of rules required to perform calculations or some other problem solving operations Especially by computer.
- Flowchart
- Pseudo code

Why Algorithms

- * Scalability:
we need to scale down a real world big prob into small steps, which helps us to analyze the problem
- * Algorithm says that each and every instruction should be followed in specific order to do a specific task.

Algorithms should consider these while creating one

- Modularity (Break problems into small chunks)
- Correctness (Generate precise output with precision)
- Maintainability (designed in simple structured way)

Some algorithmic approaches

① Brute Force algorithm:

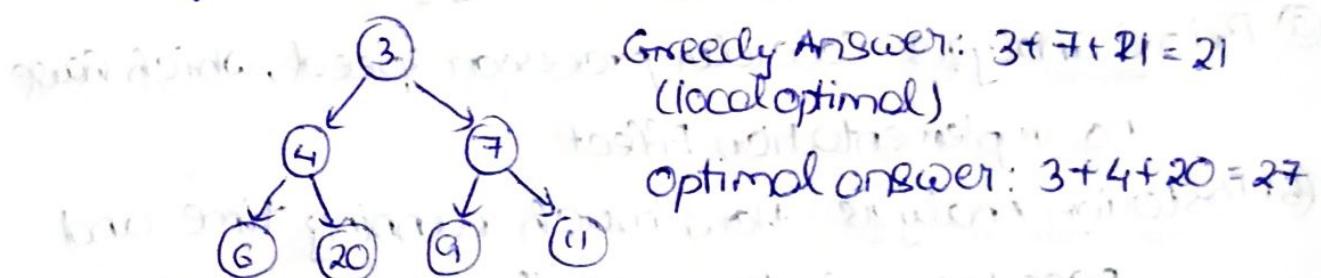
- * searches all the possibilities to provide the required solution
- Optimized method: Take best solution out of all solutions
- Sacrificing: stops at first solution, doesn't care about optimized or not

② Divide and conquer

- Divide bigger problem into smaller and solve them and merge the output's to get result of solution

③ Greedy Algorithm

- Problem solving approach of making the locally optimal choice at each stage with the hope of finding a globally optimum.
- May not provide Globally optimized values.
- One of the case that would fail



When to use?

- * Global optimum can be reached using local optimism
- * optimal solution to a problem contains opt sol of subprob

Applications

- * Activity Selection prob
- * Huffman coding
- * Job sequencing
- * Fractional knapsack
- * Prim's min spanning

④ Dynamic Programming

- * Breaks problem into subproblems
- * Stores results of subproblems using memorization
- * Find the optimal solution out of two subprob
- * Reuse the result of subproblems, to not execute twice.

⑤ Branch and bound algorithm:

- * can be applied to only integer problems
- * method of solving optimization problems by breaking them down into smaller sub problems and bound function to eliminate subproblems that cannot contain the optimal solution.

⑥ Backtracking

- * Solves the problem recursively and remove the solution if it doesn't satisfy constraints of problem.

Algorithmic Analysis

① Priori analysis: Consider processor speed, which have no implementation effect

② Posteriori Analysis: how much running time and space taken by the algorithm.

Algorithmic Complexity

① Time Complexity:

- * amount of time req to complete the execution
- * Denoted by Big(O) notation.
- * number of steps it may take to complete execution

sum=0
for i in 1 to n:

$$\text{sum} = \text{sum} + i; \quad \left\{ O(n) \right.$$

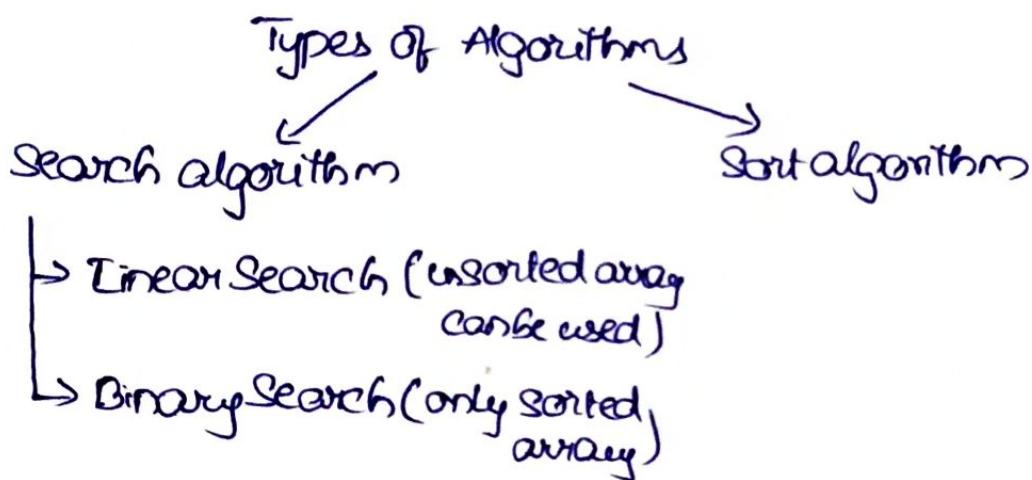
return sum; $\rightarrow O(1)$

② Space Complexity:

- * Amount of space required to solve a problem and produce an output.
- * Expressed in Big O notation.
- * Space is required by store program instructions, constant values, variable values, function calls, jumping statements etc.

Auxiliary space: Extra space required by algorithm, excluding the input size.

Space complexity: Auxiliary space + Input size



Asymptotic Analysis

we choose the best algorithm, based on time complexity.

Real time example:

If we need algorithm to add an element in front
then possible solution could be by array, Linklist.
But the best solution is through linked list because
in array's we need to push all elements to right to
add an element to the beginning array. But for
linked list we need to add a node in front and tag
the address with the second node.

- * using Asymptotic analysis, we can conclude the average case, best case and worst case scenario of an algorithm.

Example: $f(n) = 5n^2 + 6n + 12$

For small values of $n \geq 12$ is best

For large values of n (worst case) $= 5n^2$ is best

So T.C should be taken based on worst case

$n^3 \rightarrow$ More Time Complexity

$n^2 \rightarrow$ comparatively less time complexity

If we have an algo which solves in n^2 or n^3 then
we must go for n^2 as less time.C than n^3

Time Complexity

- ① worst case
- ② Average case
- ③ Best case

Asymptotic notations used for calculating T.C

* Big Oh Notation (O)

- provides the order of growth of the function
- provides an upper bound on a function
- It Ensures that program that we written will be less or equal complexity of our upper bound.
- provides worst time complexity

* Omega Notation (Ω)

- provides best case scenario
- Opposite action to Big Oh Notation
- provides lower bound of an algorithm
- provides faster time that an algorithm can run.

* Theta Notation (Θ)

- provides average case scenarios.
- provides realistic time complexity of an algorithm.

Common Asym notations

Constant	$O(1)$	→ Adding to the front of a linked list
$O(\log N)$	\log	→ Finding an entry in a sorted array
$O(N)$	linear	→ Finding an entry in an unsorted array
$O(N \log N)$	$n \log n$	→ Sorting n items by 'divide & conquer'
$O(N^2)$	quadratic	→ Shortest path b/w nodes in graph
$O(N^3)$	cubic	→ Simultaneous linear equations
$O(2^N)$	exponential	→ Tower of Hanoi problem

Pointer

- * Pointer is used to points the address of the value stored anywhere in computer memory
- * obtaining such value using pointer is called dereferencing the pointer.

Mainly useful in

① lookup tables

② Traversing String

③ control tables

④ Tree Structures

Pointer can be used in ways

1) Pointer arithmetic

$\Rightarrow ++, --, +, -$

2) Array of Pointers

3) pointer to pointer

4) passing pointer to function

5) Return pointer from function

$a \rightarrow [10] \rightarrow \text{value}$
 $2000 \rightarrow \text{address}$

$b \rightarrow []$
 3000

$b = \&a \rightarrow [] \rightarrow [10]$
 $(b \text{ points } a)$
 $3000 \rightarrow 2000$

Pointer
`int a=5
int *b;
b = &a;`

Pointer to Pointer
`int a=5;
int *b;
int **c;
b = &a;
c = &b;`

Structure

- * Structure is a composite data type that defines a grouped list of variables that are placed under one name in block of memory.
- * Allows different variables to be accessed by using a single pointer to structure.

```
struct structure_name {  
    datatype member-1;  
    datatype member2;  
    :  
    datatype member;  
};
```

```
struct structure_nm e, e2, e3;
```

Advantages

- * can hold variables of different datatypes.
- * reuse the data layout across programs.
- * used to implement LL, stacks, queues, trees, graphs etc.