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 data type?

* 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 values)
- Non Primitive Data Structure
  - Linear DS
    - Arrays
    - LinkedList
    - Stacks
    - Queues
    - (Store data in seq manner)
  - Non Linear DS
    - trees
    - Graphs
    - (1element connected with n numbers of elements)

Data structures

- Static DS (size allocated at compile time)
  Maximum size is Fixed
- Dynamic DS (size allocated at the 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 ⇒ Blueprint
DS ⇒ Implementation

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

Advantages of DS

1. Efficiency (Efficient DS → Efficient time & space)
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: Elementry value or collection of values

② Data Item: Single unit of value

③ Group Items: Data items that have Subordinate D.Item

Ex: Employee Name (First, Last, Middle Name)

④ Elementry Item: Data Item which are unable to Divide (EID)

⑤ Entity: Object that has a distinct identity (Person, Place)

⑥ Attribute: Characterstic of An Entity

Data Structures => Allows us store & organise data.

Algorithms => process the data meanigfully

* Non Premitive Data Structure are data structures derived from Primitive DS

* NPDS forms set of data elements that is either group of homogeneous / hetrogeneous Dat Structure

└ same DT form as grp   └ Diff DT form 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.

Array Elements

a =	2	6	11	7	18	4
	0	1	2	3	4	5

← Array Indices

Arrays

- 1D array (1 index)
- 2D array (2 indices)
- ND array (n indices)

Applications

① Store list of data elements of same type

② use as auxilary 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

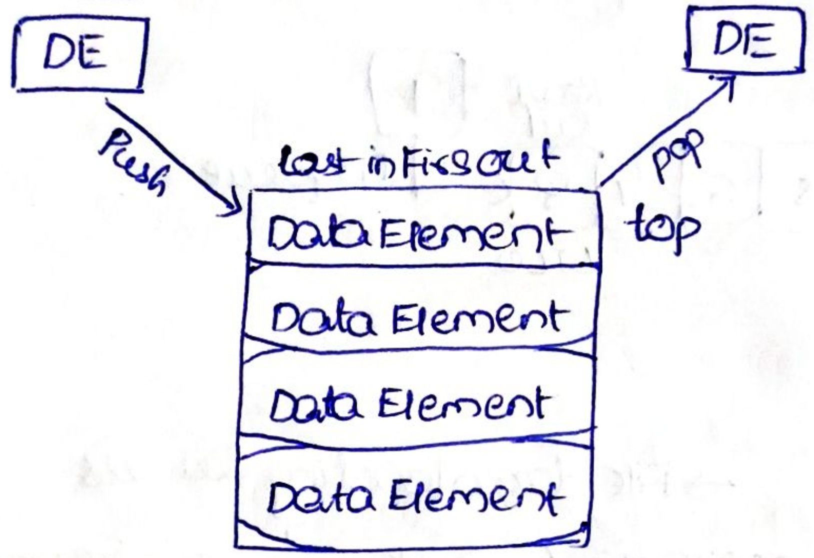
First slide → Last slide

* 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:

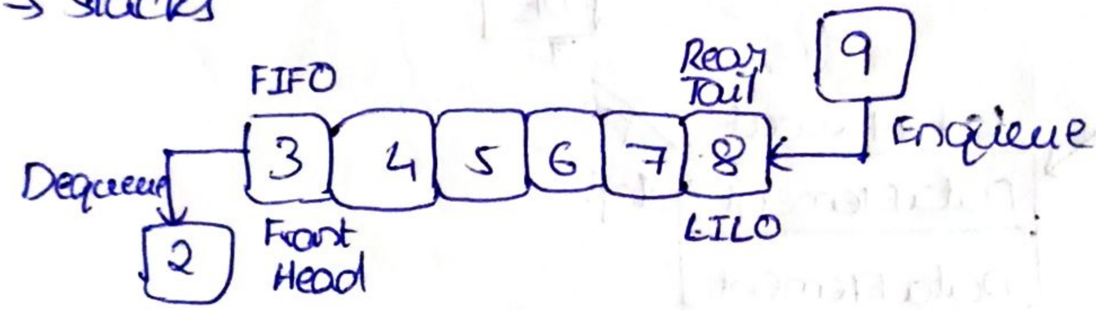
* Temporary storage for recursive operations, function calls, nested operatios
* Evaluate arithematic Expressions
* Infix Exp → Postfix Exp
* Match paranthesis
* Reverse a String
* Backtracking
* DFS in graph & tree traversal
* undo/Redo func

Queues

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

Can be implemented by
→ Arrays
→ LinkedLists
→ Stacks

Real life Examples
ticket counter
Escalator
car wash



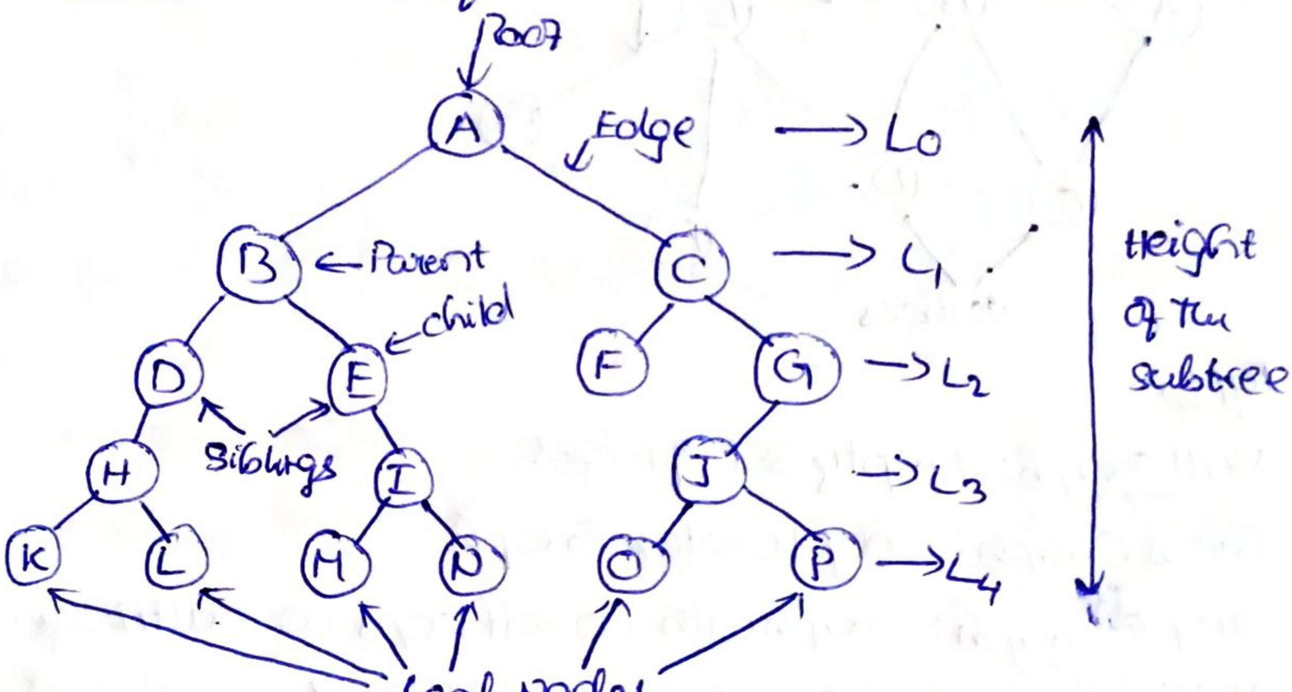
Applications
→ BFS in Graphs
→ File downloading Queues
→ Job scheduling operations
→ Handling intreuppts generated by user apps
→ CPU/Job/Disk Scheduling

Non Linear Data Structure

* Data elements not arranged in sequential order.
* Insertion & Removal is not that easy, hierachial 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 → atmost 2 children

Binary Search Tree: can maintain sorted list of nums

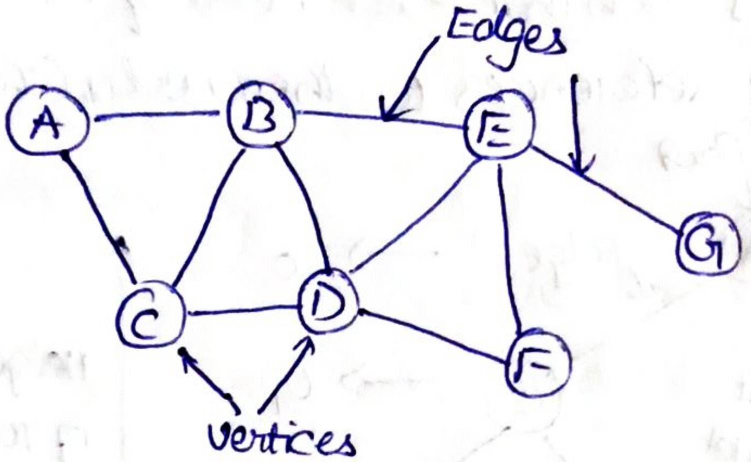
AVL Tree: self Balancing Binary search tree, Each node have Balance Factor (-1, 0, 1)

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) => set of vertices & Edges



Types

Null Graph: Empty set of Edges

Trival 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 on 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

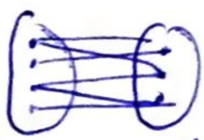
Cycle graph: Atleast 3 vertices and edges form a cycle

Cyclic graph: atleast one cycle exists

Acyclic graph: zero cycles

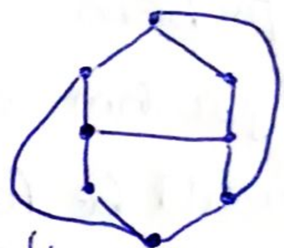
Finite / Infinite Graph: Finite / Infinite numb of vertices / Edges

Bipartite Graph: vertices can be divided into 2 sets
SetA vertices can be connected to SB 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

Basic Operations of DS

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 of LL, user don't know the implementation taken or This is main idea of Abstract Datatype

Applications of DS

→ Rep info of DB
→ search through org data
→ Generate the Data
→ Encrypt & Decrypt the data

Algorithm

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

why Algorithms

* Scalability:
  we need to scale down a real world big prob into small steps, which helps us to analyse 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 are

→ Modularity (break problems into small chunks)
→ Correctness (Generate precise output with precise inp)
→ Maintainability (designed in simple structured way)

Some algorithmic approaches

① Brute Force algorithm:

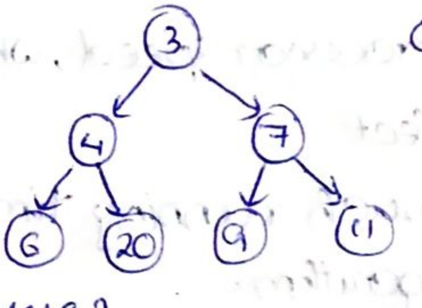
* searches all the possibilities to provide the required solution
  - → Optimezed method: Take best solution out of all solutions
  - → Sacrificing: stops at first solution, doesn't care about optimized or not

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 globaly optimum.
- → May not provide Globally optimized value
- → one of the case that would fail



Greedy Answer: $3+7+11 = 21$ (local optimal)

Optimal answer: $3+4+20 = 27$

when to use?

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

Applications

* Activity selection prob
* Huffman coding
* Job sequencing
* Fract knapsack
* Prim's Min spanning

④ Dynamic Programming

* Breaks problem into subproblems
* Stores results of sub problems using memorization
* Find the optimal solution out of thus 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 bounding function to eliminate subproblems that cannot contain the optimal solution.

6) Backtracking

* solves the problem recursively and remove the solution if it doesnt satify constraints of problem.

Algorithmic Analysis

ⓐ Priori analysis: Concider processor speed, which have no implementation Effect

ⓑ Posterior 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:
    sum=sum+i;    } O(n)
return sum;       -> 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.

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

Space complexity: Auxilary space + Input size

Types of Algorithms

- Search algorithm
  - Linear Search (unsorted array can be used)
  - Binary Search (only sorted array)
- Sort algorithm