Data Structures and Algorithms

- Difference between “Information” and “Data”

  - The quantities, characters or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical or mechanical recording media.

    Example: c = a + b ⇒ Here, a and b are data.

  - If data is arranged in a systematic way, then it gets a structure and becomes meaningful. This meaningful or processed data is called **Information**.

    Example: DATA ⇒ TEERPSAJ SI EMAN YM (just a collection of characters)
    INFORMATION ⇒ MY NAME IS JASPREET

- To provide an appropriate way to structure the data, we need to know about DATA STRUCTURES.

- A **Data Structure** is the systematic way to organize data so that it can be used efficiently.
  Example: Arrays, Linked Lists

- **Real life examples of Data Structures:**
  - Undo/Redo feature ⇒ Stack is used for this feature
  - Store an image as a bitmap ⇒ Array is used for this purpose
  - Storing friendship information on a Social Networking Site ⇒ Graph is used for this purpose

- Two important things about data types:
  - Defines a certain **domain** of values
  - Defines **Operations** allowed on those values

- **Abstract Data Types** are like user-defined data types which define operations on values using functions without specifying what is there inside the function and how the operations are performed.

  Example: Stacks

- Data structures are used to implement ADTs.
- ADT tells us **what** is to be done and data structures tell us **how** to do it.
- ADT is the **blueprint** while Data Structure is the implementation.

- Advantages of Data Structures:
  - Efficiency
  - Reusability
  - Abstraction

- A data structure is linear when all the elements are arranged in a linear (sequential) order.
  Example: Arrays, Linked Lists, Stacks, Queues

- A data structure is non-linear when all the elements are not arranged in a linear (sequential) order. There is no linear arrangement of the elements.

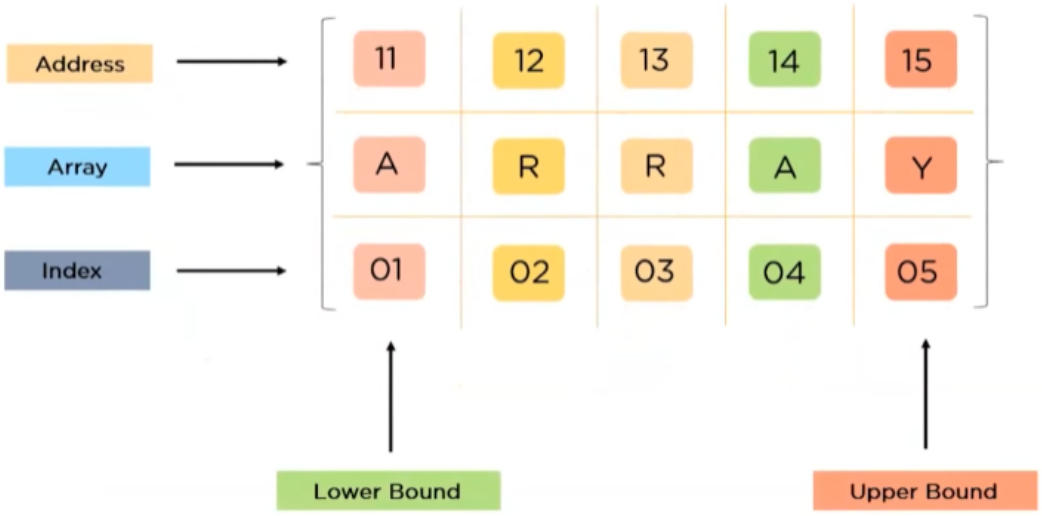
- Static Data Structures:
  - In these types of data structures, the memory is allocated at compile time. Therefore, the maximum size is fixed.

    Example: Arrays
    Advantage: Fast access
    Disadvantage: Slower insertion and deletion

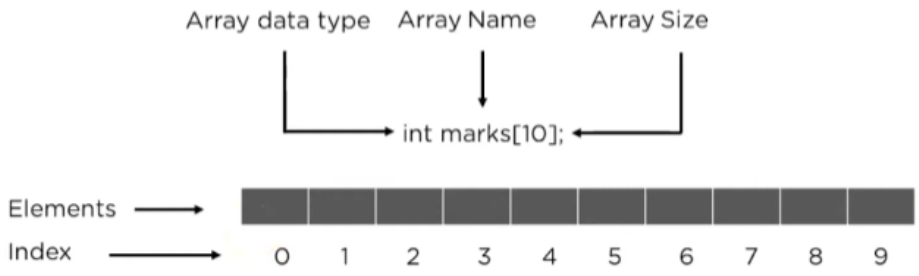
- Dynamic Data Structures:
  - In these types of data structures, the memory is allocated at run time. Therefore, the maximum size is flexible.

    Example: Linked Lists
    Advantage: Faster insertion and deletion
    Disadvantage: Slower access

- An **Array** is a linear data structure that collects elements of the same data type and stores them in contiguous and adjacent memory locations.



- Types of Arrays:
  - One-dimensional arrays : require only one subscript to specify elements in an array.



  - Multi-dimensional arrays (2D and 3D Arrays): require more than one subscript to specify elements in an array.

- **Stack:**
  - an abstract data type
  - follows LIFO (Last-In-First-Out)
  - two operations : Push and Pop

- Efficiently implemented using arrays
- An element gets inserted at the top of the stack when “push”(ed) and the top element gets removed and returned when “pop”(ed).

➙ **Queue:**
- an abstract data type
- follows FIFO (First-In-First-Out)
- Two operations : Push and Pop
- Efficiently implemented using Linked Lists
- An element gets inserted at the end of the queue when “push”(ed) and the first element gets removed and returned when “pop”(ed).

➙ **Deque:**
- An element can be inserted at the back of the queue as well as front of the queue. Similarly, an element can be removed from the back as well as the front of the queue. (It thus can be considered a stack + queue Data Structure)

➙ **Linked Lists:**
- Unlike arrays, Linked Lists are not contiguous in the memory.
- Single Linked List ⇒ Navigation is forward only
- Doubly Linked List ⇒ Forward and backwards navigation is possible
- Circular Linked List ⇒ Last element is linked to the first element.

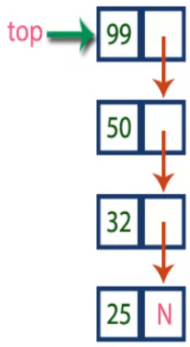
➙ **Single Linked List**
- Each node in the list consists of 2 things : A pointer and a value
- Value part of the node stores the value associated with the node.
- Pointer stores the memory address of the next node in the list.
- The first node of the list is termed as “Head” or “Root” of the list. A pointer is used to refer to the very first node of the linked list.
- The last node of the list is said to be “Tail” of the list.

➙ **Doubly Linked List**
➙ **Circular Linked List**

Linked List

searce

Use Case



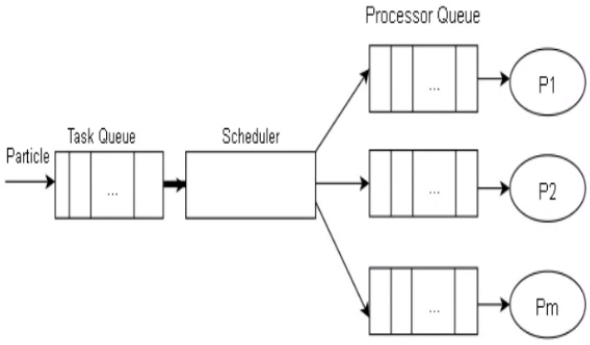
In Computer Science / Data Structures:

- Implementation of **stacks and queues**
- Implementation of graphs : **Adjacency list representation of graphs** is most popular which is uses linked list to store adjacent vertices.
- **Dynamic memory allocation** : We use linked list of free blocks.
- Maintaining **directory of names**
- Performing **arithmetic operations** on long integers
- **Manipulation of polynomials** by storing constants in the node of linked list
- **Sparse matrices** representing

Linked List

searce

Use Case



In Real World :

- **Music Player** - link songs in a play list
- **Web Browser** - visit next and previous web page
- **Image Viewer -** next and previous image
- **Operating System** - Round Robin time sharing approach for resource sharing, process execution
- **Multiplayer games** - swap between players
- Undo Redo operations
- Changing Tab in browser

Linked List

searce

Problems

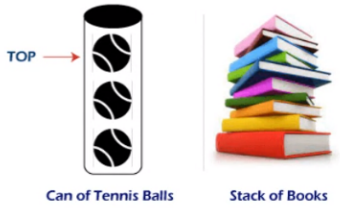
- Find nth Element / node of linked list
- Middle Element of Given List
- Count number of times element occur in list
- Length of List
- Detect Loop in Linked list

Stack

searce

What, Why and Where…

- Linear Data Structure
- LIFO(Last In First Out) or FILO(First In Last Out)
- Eg: Plates stacked over one another
- Basic operations performed in the stack :
  - Push
  - Pop
  - Peek or Top
  - isEmpty
- Time Complexity for all operations : O(1)
- Two ways to implement Stack :
  - Array
  - Linked List
- Overflow, Underflow Conditions



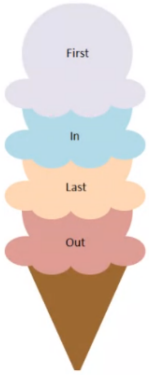
Stack

searce

What, Why and Where…

Applications of stack:

- Balancing of symbols
- Infix to Postfix /Prefix conversion
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
- Backtracking
- In Graph Algorithms like Topological Sorting and Strongly Connected Components
- In Memory management
- String reversal



Stack

searce

Implementations

- Implement Queue using Stack
- Implement stack using Queue
- Design stack to getMin() in O(1) time
  - With O(n) extra space
  - With O(1) extra space

Stack

searce

Problems using Stack and Operations on Stack

- Infix to postfix
- Prefix to Postfix
- Postfix to Infix
- Next greater number
- Expression evaluation

- Reverse Stack using recursion
- Stack sorting
- Sort array using stack

Queue

searce

What, Why and Where ?

- **Breadth First Search**
- When a **resource is shared among multiple consumers**. Examples include CPU scheduling, Disk Scheduling.
- When **data** is **transferred asynchronously** (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.
- **In Operating systems:**
  a) Semaphores
  b) FCFS (first come first serve) scheduling, example: FIFO queue
  c) Spooling in printers
  d) Buffer for devices like keyboard
- **In Networks:**
  a) Queues in routers/ switches
  b) Mail Queues
- **Variations:** (Deque, Priority Queue, Doubly Ended Priority Queue)

Queue

searce

Problems

- Sliding window - maximum of subarrays of size k
- Level order traversal
- BFS for a Graph
- Check if all levels of two trees are anagrams or not

➙ **Hashing**

- Used to index large amounts of data.
- Address of each key is calculated using the key itself.
- Collisions can be resolved with either Open addressing or Closed addressing.
- In the best case scenario Insertion, Deletion and Retrieval take O(1) time. In case of collisions, this will not hold true.
- Widely used in database indexing, compilers, caching, password authentication, network (routers, server), substring search, string commonalities, file/directory synchronization and more.
- Collision Resolution techniques:
  - Open Addressing
    - Linear probing
    - Plus 3 rehash
    - Quadratic Probing (failed attempts)^2
    - Double Hashing
  - Closed Addressing
- Objectives of Hash Function
  - Minimize collisions
  - Uniform distribution of values
  - Easy to calculate
  - Resolve any collisions

➙ Hashing in Python

- Dictionary:
  - An Abstract Data Type (ADT)
  - Maintains set of items where each item has a key
  - We can insert and delete an item
  - When inserting an item, if the key already exists, it overwrites the existing value
  - We can search for a key. If the key doesn't exist, report key error
- Data type : dict()
  - D[key] ⇒ search
  - D[key] = value ⇒ insert
  - del D[key] ⇒ delete
- An item in Python dictionary is a pair of key and value, i.e., (key, value)

➙ Approaches for Hashing

- Simple approach:
  - Direct access table:
    - Store items in an array indexed by key
    - Issues:
      - ***First*** : Keys must be an integer and must be non-negative values
      - ***Second*** : Gigantic memory hog
    - Solution to ***First*** :
      - Prehash:
        - Maps keys to non-negative integers
        - In theory, keys are finite and discrete
        - In Python, hash(x) is the prehash(x)
        - Issues like hash('\0B') = hash('\0\0C')
        - Ideally, if hash(x) = hash(y), x and y must hold x==y
    - Solution to ***Second*** :
      - Hashing:

- Reduce universe U of all keys (integers) down to reasonable size **m** for table
- h : U ⇒ {0, 1, ……., m-1}

- Chaining
  - Each index of the table is a head pointer to a linked list
  - In case of collision, the object is simply appended at the end of the linked list corresponding to the index

- Hash functions
  - **Division method**: h(k) = k mod m ⇒ pretty good if m is prime and m is not close to a power of 2 and power of 10
  - **Multiplication method**: h(k) = [(a*k) mod 2^w] >> (w-r)
    w ⇒ bit-size of our machine
    a ⇒ random integer among all possible w-bit integers, better be odd and should not be very close to power of 2 and between 2^{r-1} and 2^r
    m ⇒ 2^r
  - **Universal hashing**: h(k) = [(ak+b) mod p] mod m
    p ⇒ prime number > |U|
    a, b ⇒ random number in range [0, p-1]

    For worst-case keys, $k_1 \neq k_2$: $P(h(k_1)=h(k_2)) = 1/m$

➙ Queues

- Principle : FIFO (First-In-First-Out)
- Terminologies :
  - Insertion ⇒ Enqueue
  - Deletion ⇒ Dequeue
  - An element get inserted from the **REAR** end and deleted from the **FRONT** end
- Operations :
  - Enqueue ⇒ To insert an element in the queue
  - Dequeue ⇒ To delete an element from the queue
  - Front/Peek ⇒ To get the element present at the front of the queue without deleting the element from the queue

    - isFull ⇒ To check whether the queue is full or not. Returns boolean value.
    - isEmpty ⇒ To check whether the queue is empty or not. Returns boolean value.
- Applications :
    - It is used when we want to serve a request on a single shared resource
    - Processors