Summary of Data Structure

Data Structure:

Data structures refer to data and representation of data object within program, which means implementation of structured relationship.

Data structure is collection of atomic and composite data type in a set with defined relationship.

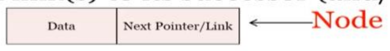
The best suitable definition for Data structure is Organizing data, managing data and store the data in appropriate format so that data could be access and modify efficiently.

Linear Data structure

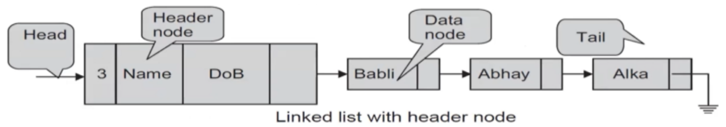
- Data structure is said to be linear if its elements form a sequence or linear list.
- Linear Data structure, every data element has unique successor and unique predecessor.
- Example of Linear structure are **array, stack, queue, linked list.**
- A linear search is slower than binary search.

Linked Lists:

- Linear, Dynamic data structure
- linked lists do not necessarily contain consecutive memory locations
- To maintain the specific sequence of these data items, we need to maintain link(s) with a successor
- A linked list is an ordered collection of data in which each element (node) contains a minimum of two values, data and link(s) to its successor (and/or predecessor).

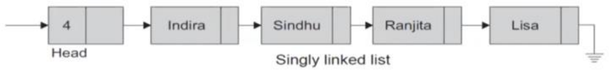


- Tail pointer: pointer pointing to the last node of a linked list called the tail pointer.

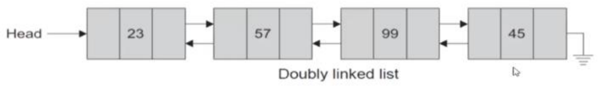


Linked list with header node

- Thus, linked lists can be classified broadly as follows:
  1. Singly linked list(SLL)
  2. Doubly linked list (DLL)



Singly linked list



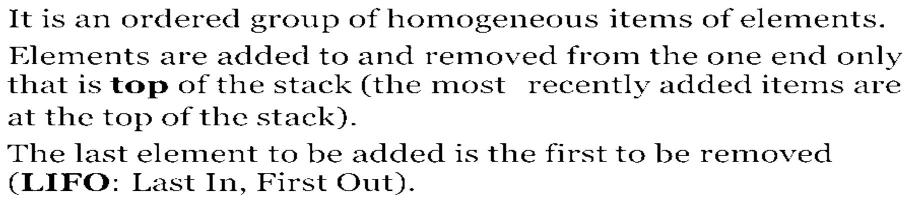
Doubly linked list

Apart from these terms in LinkedList we should know, In a Linked List, the values can be of any type. Linked list can be of two types one is singly and doubly linked list. In Singly link list, node contains only two fields one for data and second for link to successor but in doubly LinkedList, node will have three fields. One for link to predecessor, second for data and third for link to successor. A doubly linked list can be a good choice for a Queue implementation and in a doubly linked list, the direction of traversal can change. One of the benefits of Linked list is, it allows efficient insertion or removal of elements from any position during iteration.

We can perform Insertion, Traversal and Deletion on a Singly Linked List.

Stacks & Queues:

Stack:



It is an ordered group of homogeneous items of elements.

Elements are added to and removed from the one end only that is **top** of the stack (the most recently added items are at the top of the stack).

The last element to be added is the first to be removed (**LIFO**: Last In, First Out).

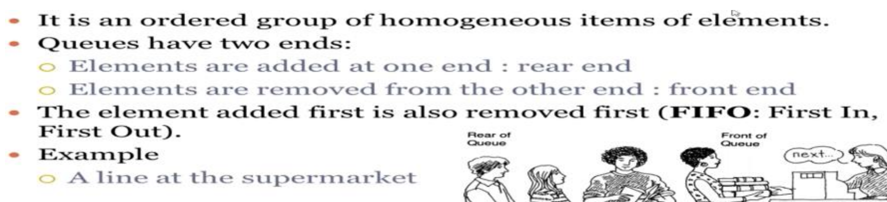
But apart from this, stack can be of fixed size, It cannot be empty initially and using linked list we can implement it. As we have seen stack follows first in last out. If we will have to understand it very well then, we will take an example. Assume you are working on a project and writing some code in VS code. You are done with your code but now you want to go back to previous version (Don't want newly added lines of code) so for undoing we will press **ctrl+z,** which is doing undo step by step which can be very good example of stack. Means code which we wrote last will be undone first.

Second example we can consider is, you are developing a web browser and you will have to store a history of your browsing to go back and forward on button click. So, for storing history we can use stack. Queue can't use in such kind of scenarios.

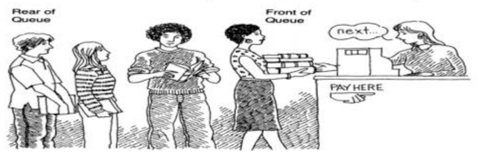
There are three common operations in Stack

1. **Push** → to insert element in stack
2. **POP** → to delete top element in stack
3. **Peek** → to get value at the top of a stack without removing

Queue:



- It is an ordered group of homogeneous items of elements.
- Queues have two ends:
  - Elements are added at one end : rear end
  - Elements are removed from the other end : front end
- The element added first is also removed first (**FIFO**: First In, First Out).
- Example
  - A line at the supermarket



In Queue, we have seen “A Queue is a linear data structure.”

1. **Enqueue:** Insert element from rear
2. **Dequeue:** Remove element from front

Sorting and searching array data:

Bubble sort algorithm → Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

- Bubble sort is not very complex and very efficient too.
- As we have seen in bubble sort we have passes = number of elements -1 while insertion sort may have more passes than bubble sort.

Insertion sort algorithm →

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Characteristics of Insertion Sort:

This algorithm is one of the simplest algorithms with simple implementation

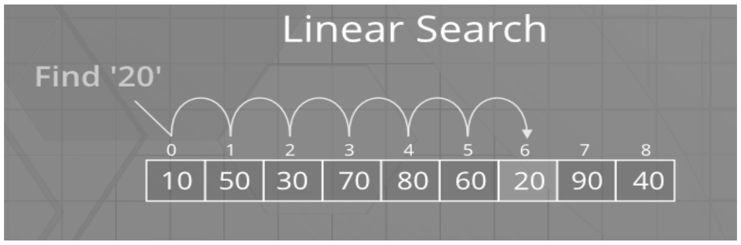
Basically, Insertion sort is efficient for small data values

Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.

- Key points to keep in mind while working with Insertion algorithm.
    - Insertion sort divide our array in two sub arrays. One is called as sorted array and second is called as unsorted array.
    - We can use insertion sort to sort data, this data can be coming from any source it may be received from internet too.
    - We have seen insertion sort and it’s working so, we concluded that insertion sort is not good choice when data which is supposed to sort is very large.

Linear search algorithm:

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. It is the easiest searching algorithm.



In Linear search, the target value is compared to the adjacent element in the array.

Binary Search algorithm →

- As we have seen Linear Search which is having worst time complexity so, in case of worst cases we can use binary search algorithm.
- **Binary Search Approach:**
    - *Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).*
- **<u>Binary Search Algorithm:</u>** The basic steps to perform Binary Search are:
    - Begin with the mid element of the whole array as a search key.
    - If the value of the search key is equal to the item then return an index of the search key.
    - Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
    - Otherwise, narrow it to the upper half.
    - Repeatedly check from the second point until the value is found or the interval is empty.
    - In simple words, we can say binary search uses Recursive method for implementation.
    - Most important point that you should keep in mind is Binary search can implement only on sorted array.

Sets:

- A set is a data structure that stores unique elements of the same type in a sorted order. Each value is a key, which means that we access each value using the value itself. With arrays, on the other hand, we access each value by its position in the container (the index). Accordingly, each value in a set must be unique.
- A set is a well-defined collection of objects. The objects may be numbers, alphabets, names of people, etc. Sets are represented using upper-case letters such as A, B, etc.

- We can implement Set using any programming language too. Set is itself is collection of unique elements, we don't have any function for filtering data and get unique values. Like in SQL we used DISTINCT keyword to get unique values but in SET we don't have such function in set.

For example:

- A = {a, e, i, o, u}
- A is a set of vowels in the English alphabet

Set operations:

Intersection:

- The intersection of two sets A and B is a set that contains all the elements that are common to both A and B. Formally it is written as
- It is representing pictorial way as: A Ω B
- *If A = {2, 3, 5, 7} and B = {1, 2, 3, 4, 5}*
  then the intersection of set A and B is the set A ∩ B = {2, 3, 5}
- *In this example 2, 3, and 5 are the only elements that belong to both sets A and B.*

Union:

- Union of two sets A and B is a set that contains all the elements that are in A or in B or in both A and B. Formally it is written as A U B.
- Pictorially ii is represented as:
- *If A = {2, 4, 8} and B = {2, 6, 8}*
  then the union of A and B is the set A ∪ B = {2, 4, 6, 8}

In this example, 2, 4, 6, and 8 are the elements that are found in set A or in set B or in both sets A and B. A set of all distinct items which exist within any of the input sets is called union.

Some topics we have seen on very first day. Kindly go through ppt of first day too.

Structured Data in Aggregate Data Type

Benefits of Structured Data:

Avoid one-off errors - In languages that have zero-based indices, it is common for programmers to try to access memory outside the bounds of the aggregate data type. Know your limits; use sizeof()!

- Use structured data to make the relationships between data clear.
- Use structured data to make operations of blocks of data simpler.
- Use structured data to make parameter passing clearer, easier and simpler.

Abstract Data Types (ADTs)

An abstract data type is a collection of data and operation that work on that data.

Benefits of Using Abstract Data Type:

- Support information hiding
- Minimize the effect of changes in program.
- Improve ability to enhance code
- Program becomes more self-documenting
- Allows for object-oriented concepts
- It defines operations dealing with real-world entities

Kindly go through this file, all topics we have seen during DS training, those topics I put in this file so, read very carefully. Revise ppts also.