Data structures:

**1.Linked List**:

* A linked list is a data structure that consists of sequence of nodes. Each node is composed of two fields: data field and reference field which is a pointer that points to the next node in the sequence. Linked List Node.
* This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.
* more efficient insertion or removal of nodes. The list elements can be easily inserted or removed
* A drawback of linked lists is that access time is linear (and difficult to [pipeline](https://en.wikipedia.org/wiki/Instruction_pipelining)). Faster access, such as random access, is not feasible. [Arrays](https://en.wikipedia.org/wiki/Array_data_structure) have better [cache locality](https://en.wikipedia.org/wiki/Locality_of_reference) compared to linked lists.

**Types of Linked Lists:**

* singly linked list: It is the most common. Each node has data and a pointer to the next node.
* doubly linked list: We add a pointer to the previous node in a doubly linked list. Thus, we can go in either direction: forward or backward.
* Circular linked list: A circular linked list is a variation of linked list in which the last element is linked to the first element. This forms a circular loop. A circular linked list can be either singly linked or doubly linked.

Advantages:

* **Dynamic Data Structure**
* **Insertion and Deletion:** Insertion and deletion of nodes are really easier. Unlike array here we don’t have to shift elements after insertion or deletion of an element.
* **No Memory Wastage**
* **Implementation:** Data structures such as stack and queues can be easily implemented using linked list.

Disadvantages:

* **They use more memory than**[**arrays**](https://en.wikipedia.org/wiki/Array_data_structure)**because of the storage used by their**[**pointers**](https://en.wikipedia.org/wiki/Pointer_(computer_science))**.**
* **Nodes in a linked list must be read in order from the beginning as linked lists are inherently**[**sequential access**](https://en.wikipedia.org/wiki/Sequential_access)**.**
* **Nodes are stored in contiguously, greatly increasing the time periods required to access individual elements within the list, especially with a**[**CPU cache**](https://en.wikipedia.org/wiki/CPU_cache)**.**
* **Difficulties arise in linked lists when it comes to reverse traversing.**

2. **Queue:**

* a line or sequence of people or vehicles awaiting their turn to be attended to or to proceed
* A queue is a useful data structure in programming. Queue follows the **First In First Out(FIFO)**rule - the item that goes in first is the item that comes out first too.
* A queue is an object or more specifically an abstract data structure(ADT) that allows the following operations:
  + Enqueue: Add element to end of queue
  + Dequeue: Remove element from front of queue
  + IsEmpty: Check if queue is empty
  + IsFull: Check if queue is full
  + Peek: Get the value of the front of queue without removing it
* In queues insertion can take place at only one end called rear.
* In queues deletions can takes place at the other end called front.
* Queues are called FIFO(first in first out). The element first into the queue is the element deleted first from the queue.Queues are also called LILO(last in last out).The element entered last into the queue is the element deleted last from the queue.
* In programming terms, putting an item in the queue is called an "enqueue" and removing an item from the queue is called "dequeue".

Different types of queue:

            1. Abstract queue

            2. Queue

            3. Priority queue

            4. Circular queue

            5. De queue.

**3. Stack:**

* **Last In First Out** - the last item that was placed is the first item to go out.
* In programming terms, putting an item on top of the stack is called "push" and removing an item is called "pop".
* A stack is an object or more specifically an abstract data structure(ADT) that allows the following operations:
* Push: Add element to top of stack
* Pop: Remove element from top of stack
* IsEmpty: Check if stack is empty
* IsFull: Check if stack is full
* Peek: Get the value of the top element without removing it
* stack is a simple data structure to implement, it is very powerful. The most common **uses** of a stack are:
* **To reverse a word** - Put all the letters in a stack and pop them out. Because of LIFO order of stack, you will get the letters in reverse order.
* **In compilers**
* **In browsers**.
* Stacks provide a unique way to work with contiguous memory. Very similar to Arrays and Lists, Stacks provide a way for users to access different pieces of contiguous data in a *Last In First Out* manner.
* Easy to get started

Advantages:

\* Low Hardware Requirement

\* Cross- Platform

\* Anyone with access can edit the program

### Disadvantages:

### \* Inflexible

### \* Lack of scalability

### \* Unable to Copy & Paste

### Applications of stack:

### Expression evaluation and syntax parsing

### Backtracking

### Compile time memory management

4. **Hash Table:**

* A hash table (hash map) is a data structure which implements an associative array abstract data type, a structure that can map keys to values.
* A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.
* Hash Table is a data structure which stores data in an associative manner.
* In a hash table, data is stored in an array format, where each data value has its own unique index value.
* Access of data becomes very fast if we know the index of the desired data.

Basic Operations:

Following are the basic primary operations of a hash table.

* **Search** − Searches an element in a hash table.
* **Insert** − inserts an element in a hash table.
* **delete** − Deletes an element from a hash table.
* Hashing is a technique to convert a range of key values into a range of indexes of an array

Hash [collisions](https://en.wikipedia.org/wiki/Collision_(computer_science)) are:

1. Separate chaining
2. Open addressing(This method is also called closed hashing)
3. Dynamic resizing

Advantages:

* The main advantage of hash tables over other table data structures is speed.
* This advantage is more apparent when the number of entries is large.
* Hash tables are particularly efficient when the maximum number of entries can be predicted in advance, so that the bucket array can be allocated once with the optimum size and never resized.
* If the set of key-value pairs is fixed and known ahead of time (so insertions and deletions are not allowed),

Drawbacks:

* Hash tables are not effective when the number of entries is very small.
* For certain string processing applications, such as [spell-checking](https://en.wikipedia.org/wiki/Spell_checker), hash tables may be less efficient than [tries](https://en.wikipedia.org/wiki/Trie), [finite automata](https://en.wikipedia.org/wiki/Finite_automata), or [Judy arrays](https://en.wikipedia.org/wiki/Judy_array).
* Also, if there are too many possible keys to store that hash tables become quite inefficient when there are many collisions.

5. **Heap**:

* Heap Data Structure is generally taught with Heapsort. Heap Implemented priority queues are used in Graph algorithms like Prim's Algorithm and Dijkstra's algorithm.
* The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.
* The node at the "top" of the heap (with no parents) is called the *root* node.

Basic operations:

1. Insert
2. Find-max
3. Extract-max
4. Delete max
5. Replace

The heap data structure has many Applications:

* [Heapsort](https://en.wikipedia.org/wiki/Heapsort)
* [Selection algorithms](https://en.wikipedia.org/wiki/Selection_algorithm)
* [Graph algorithms](https://en.wikipedia.org/wiki/List_of_algorithms#Graph_algorithms)
* [Priority Queue](https://en.wikipedia.org/wiki/Priority_Queue)
* [K-way merge](https://en.wikipedia.org/wiki/K-way_merge_algorithm)
* A heap is a complete binary tree, it has a smallest possible height
* A heap is a specialized [tree](https://en.wikipedia.org/wiki/Tree_(data_structure))-based [data structure](https://en.wikipedia.org/wiki/Data_structure) that satisfies the *heap* property
* The heap is one maximally efficient implementation of an [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) called a [priority queue](https://en.wikipedia.org/wiki/Priority_queue).
* A heap is a useful data structure when you need to remove the object with the highest (or lowest) priority.

Use of heap:

* The first is as a way of implementing a special kind of queue, called a priority queue
* The second application is sorting

Complexity of heapsort Algorithm:

* Each insert and delete operation is O(logN)

6. **Priority Queue**:

* A **priority queue** is an [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) which is like a regular [queue](https://en.wikipedia.org/wiki/Queue_(abstract_data_type)) or [stack](https://en.wikipedia.org/wiki/Stack_(abstract_data_type)) data structure, but where additionally each element has a "priority" associated with it.
* In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

Basic operations are:

* insert / enqueue − add an item to the rear of the queue.
* remove / dequeue − remove an item from the front of the queue.

**Applications of Priority Queue:**  
1) CPU Scheduling  
2) Graph algorithms like [Dijkstra’s shortest path algorithm](https://www.geeksforgeeks.org/greedy-algorithms-set-7-dijkstras-algorithm-for-adjacency-list-representation/), [Prim’s Minimum Spanning Tree](https://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-mst-for-adjacency-list-representation/), etc  
3) All [queue applications](https://www.geeksforgeeks.org/applications-of-queue-data-structure/) where priority is involved.

There are two types of priority queues:

* 1. Max priority queue
  2. Min priority queue

7.**Tree:**

* A tree is a widely used [Abstract Data Type](https://en.wikipedia.org/wiki/Abstract_data_type) (ADT) or [data structure](https://en.wikipedia.org/wiki/Data_structure) implementing this ADT.
* A tree data structure can be defined [recursively](https://en.wikipedia.org/wiki/Recursion) (locally) as a collection of [nodes](https://en.wikipedia.org/wiki/Node_(computer_science)) (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes.
* A tree is a data structure made up of nodes or vertices and edges without having any cycle.
* The tree with no nodes is called the null or empty tree.
* A tree that is not empty consists of a root node and potentially many levels of additional nodes that form a hierarchy.

Common uses:

* Representing [hierarchical](https://en.wikipedia.org/wiki/Hierarchical) data
* Storing data in a way that makes it efficiently [searchable](https://en.wikipedia.org/wiki/Search_algorithm) (see [binary search tree](https://en.wikipedia.org/wiki/Binary_search_tree) and [tree traversal](https://en.wikipedia.org/wiki/Tree_traversal))
* Representing [sorted lists](https://en.wikipedia.org/wiki/Sorting_algorithm) of data.

The tree classified into many types. That are:

1. **Binary Tree**: This is the most basic basic from of tree structure. Where each node can have utmost two children. A **perfect** binary tree is a binary tree in which all interior nodes have two children and all leaves have the same depth or same level.
2. **Binary search tree:** BST is a binary tree with certain properties such as, and left child of the given node contains value less than equal to the given node and right hand child contain node greater than the given node.
3. **Red-Black tree**: Another variant of binary tree similar to AVL tree it is a self balancing binary search tree. In this tree nodes are either colored red or black.
4. **Avl tree:** AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.etc

**8. Graph**:

* Graph is a data structure that consists of following two components:  
    1. A finite set of vertices also called as nodes.  
   **2.** A finite set of ordered pair of the form (u, v) called as edge.
* Graphs are used to represent many real-life applications, Graphs are used to represent networks
* A [graph](https://en.wikipedia.org/wiki/Graph) or [chart](https://en.wikipedia.org/wiki/Chart) or [diagram](https://en.wikipedia.org/wiki/Diagram) is a diagrammatical illustration of a set of [data](https://en.wikipedia.org/wiki/Data).
* If the graph is uploaded as an image file, it can be placed within articles [just like any other image](https://en.wikipedia.org/wiki/Wikipedia:Picture_tutorial).
* Graphs must be accurate and convey information efficiently.
* They should be viewable at different computer screen resolutions.
* Types of graph are:

1 Pie graph

2 Bar graph etc.

9. **Disjoint set**

* In mathematics, two sets are said to be disjoint sets if they have no element in common. Equivalently, disjoint sets are sets whose intersection is the empty set. For example, {1, 2, 3} and {4, 5, 6} are disjoint sets, while {1, 2, 3} and {3, 4, 5} are not.
* A **disjoint**-**set** data structure is a data structure that keeps track of a **set** of elements partitioned into a number of **disjoint**(non-overlapping) subsets.
* A disjoint set data structure consists of two important functions:
* Find() - It helps determine which subset a particular element belongs to. It also helps determine if the element is in more than one subset.
* Union() - It helps check whether a graph is cyclic or not.
* The efficiency of an algorithm sometimes depends on the data structure that is used. An efficient data structure, like the disjoint-set-union, can reduce the execution time of an algorithm.
* Operations on set:

Make-set(x)

Union(x,y)

Find-set(x)

Insert

Delete

Split(a,s)

Minimum(s)

* Two application examples:

1 Connected components (CCs)

2 Minimum Spanning Trees (MSTs)

* Example:

If set A{1,2,3} and set B{3,4,5} – this is not a disjoint set. Because they have one comman{1}.

Another example is: set A{1,2,3} and B{5,6}. This is disjoint set. Because there is no comman value.