**Data structure:** In computer science, a data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

**Examples:** Linked Lists, Stacks, Queues, Hash Tables, Trees, Heaps, Graphs

**Array:** An array is a systematic arrangement of similar objects, usually in rows and columns.

**Linked Lists:** A linked-list is a sequence of data structures which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link.

**Stacks:** Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO (First In Last Out).

**Queue:** The queue is a linear data structure used to represent a linear list. It allows insertion of an element to be done at one end and deletion of an element to be performed at the other end.

**Hash Tables:** A hash table is a data structure that is used to store key-value pairs. Just like anything in life, when something seems so simple, there is probably a lot of discipline and hard work behind it.

**Tree:** A tree is a hierarchical data structure defined as a collection of nodes. Nodes represent value and nodes are connected by edges. A tree has the following properties: The tree has one node called root.

**Heap:** Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If α has child node β then − key(α) ≥ key(β) As the value of parent is greater than that of child, this property generates Max Heap.

**Graph:** A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as, A Graph consists of a finite set of vertices (or nodes) and set of Edges which connect a pair of nodes.

**STACK**

**[Last In First Out]**

🡺 Collection of objects accessed from one end

🡺 Two primitives operations

* Push: for addition
* Pop: for deletions

🡺 Constraint: All operations on TOP of stack

🡺 OVERFLOW: When Stack is FULL

🡺 UNDERFLOW: When Stack is EMPTY

**QUEUE**

**[First In First Out]**

🡺 Collection of objects

* Add to back (queue)
* Remove from front (dequeue)

🡺 Two primitive operations

* QUEUE: for addition
* DEQUEUE: for deletion

🡺 Queue FULL: When back pointer points last element

🡺 Queue EMPTY: front = back = -1 OR front > back

**LINKED LIST**

[Collections of NODES]

🡺 Collection of objects LINKED together in a SEQUENCE

🡺 Advantages:

* No fixed size: don’t need to declare the size
* Additions of elements at any position

🡺 Nodes have two parts:

* Data
* Link

🡺 Random allocation

🡺 Traversing: Navigating to a node and Random access not possible

🡺 Appending: ADD a new node to the END of list

1. Create a new Node
2. Make the next ptr NULL
3. Traverse to the last node
4. Point last node’s next to new node
5. Done

🡺 Prepending: ADD a new node to the FRONT of list

1. Create a new Node
2. Point next ptr to FIRST node
3. Point head to new node
4. Done

🡺 ADD a new node Position specified by INDEX

1. Traverse till the desired node N
2. Create a node X
3. Link X to N’s neighbor (X.next = N.next)
4. Link N to X (N.next=X)
5. Done

🡺 DELETE a node:

1. Traverse till the preceding node N
2. Link N to neighbor’s neighbor N.next = N.next.next
3. Done
4. For HEAD node head = head.next

Tree

🡺 Tree represents the nodes connected by edges.

🡺 Binary Tree is a special datastructure. Each nodes have maximum of two children.

🡺 Important terms:

* **Path** − Path refers to the sequence of nodes along the edges of a tree.
* **Root** − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
* **Parent** − Any node except the root node has one edge upward to a node called parent.
* **Child** − The node below a given node connected by its edge downward is called its child node.
* **Leaf** − The node which does not have any child node is called the leaf node.
* **Subtree** − Subtree represents the descendants of a node.
* **Visiting** − Visiting refers to checking the value of a node when control is on the node.
* **Traversing** − Traversing means passing through nodes in a specific order.
* **Levels** − Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
* **keys** − Key represents a value of a node based on which a search operation is to be carried out for a node.

🡺 Basic operations:

* **Insert** − Inserts an element in a tree/create a tree.
* **Search** − Searches an element in a tree.
* **Preorder Traversal** − Traverses a tree in a pre-order manner.
* **Inorder Traversal** − Traverses a tree in an in-order manner.
* **Postorder Traversal** − Traverses a tree in a post-order manner.