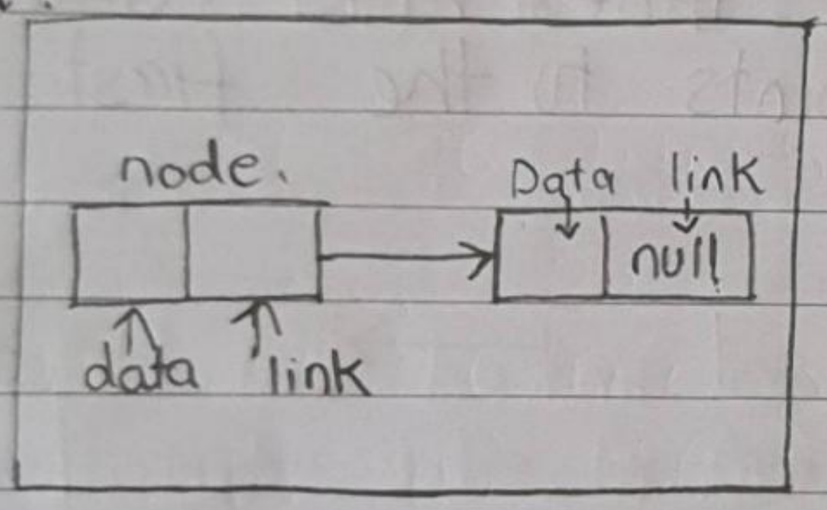
8/03/22

CHAPTERNNO : 02. LINKED LIST

* Linked List.

A linked list is a ordered collection of noteg in which each notle contains data And link to it's sucessor. Each node of the list has two elements.

1) The data being stored in the list.
2) The Pointer to the next node in the list
3) The last node in the list contain a null pointer to indecate that it is the end of the list.

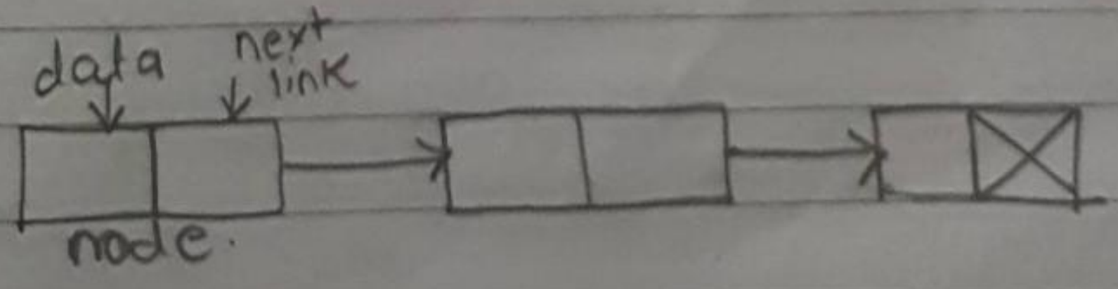


Types. The Linked List is classified as :-

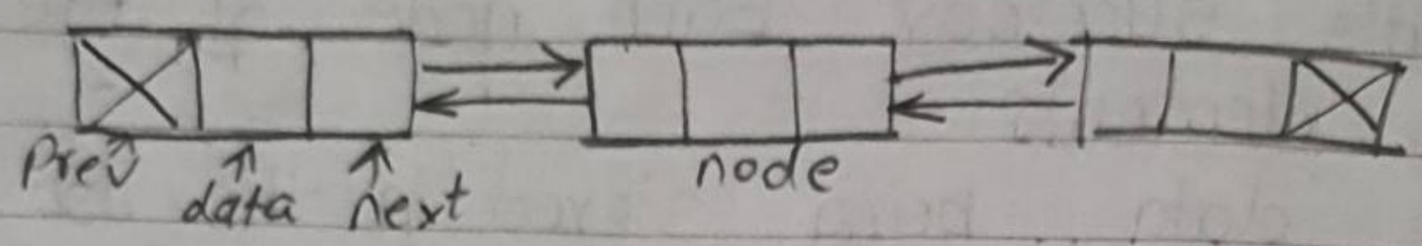
1) Singly
2) Doubly
3) Circular (Singly)
   Circular (Doubly).

1) Singly list.

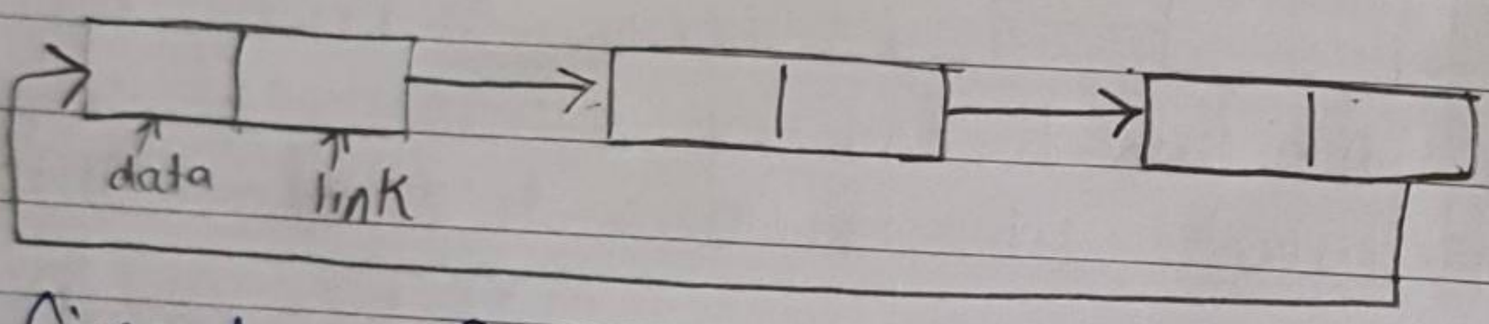
Each node of the list contains only one pointer which points to the next node of the list.



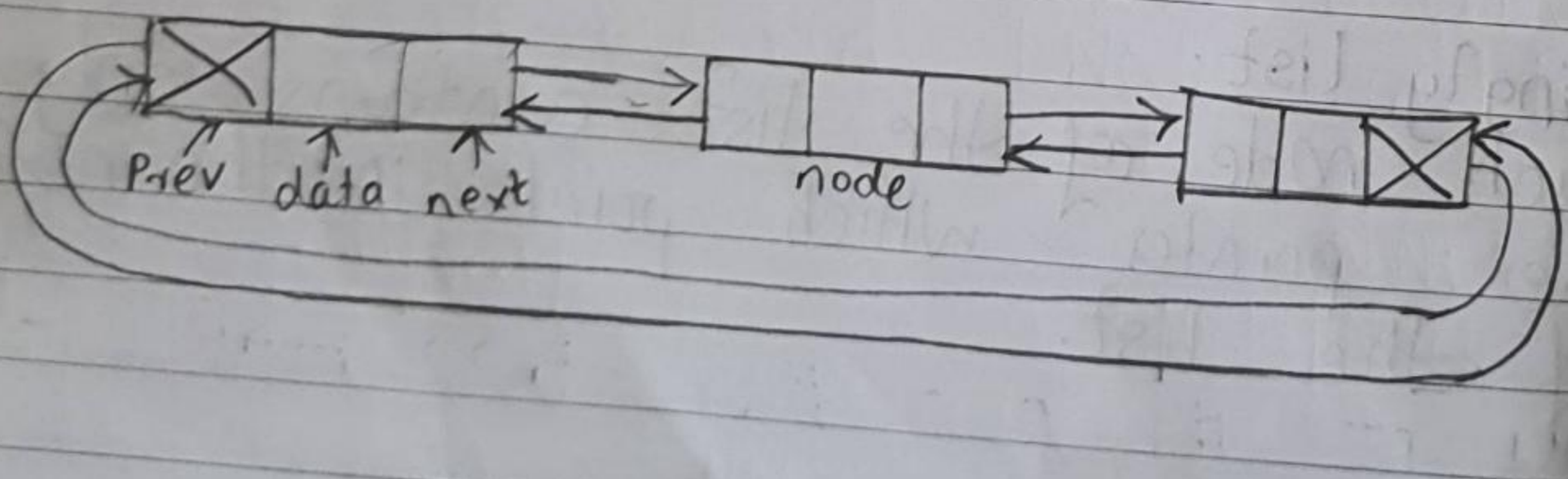
2) Doubly Linked list:-
Each node in this list contains 2 pointers one pointer to the previous node and another pointer to the next node.



3) Circular (Singly)
This is a singly linked list in which the last node does not contain a null pointer but points to the first node



4) Circular (Doubly)
It is a doubly linked list in which the last node points to the first node and first node points to the last node.



* Static Representation of Linked List.

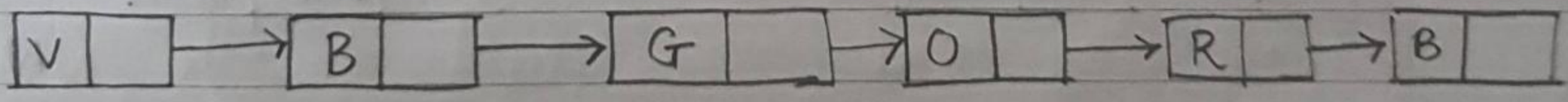
	Data
0	Blue
1	Red
2	
3	Violet
4	Green
5	Black
6	Orange

	Next/Link.
4	
5	
.	
0	
6	
-1	
1	

Data[3] = Violet
Link [3] = 0
Data [0] = Blue
Link [0] = 4
Data [4] = Green
Link [4] = 6
Data [6] = Orange
Link [6] = 1
Data [1] = Red
Link [1] = 5
Data [5] = Black
Link [5] = -1

In this Representation an Array is used to store the element of the list. the elements can be stored in the array at any position that is they are not stored sequentially. The elements are linked to one another by storing the location of the next element in another array called Link.

* Dynamic Representation
In this each element and link are linked using pointer thus the memory has to be allocated for
1) Data element.
2) Pointer to the next element.



* Creating a Node :-

```
struct node
{
    int data;
    struct node *next;
};
struct node * head , *ptr;
ptr = (struct node *) malloc (size of (struct
        node *));
```

starts pointer of location of memory

* Operations on Linked list.

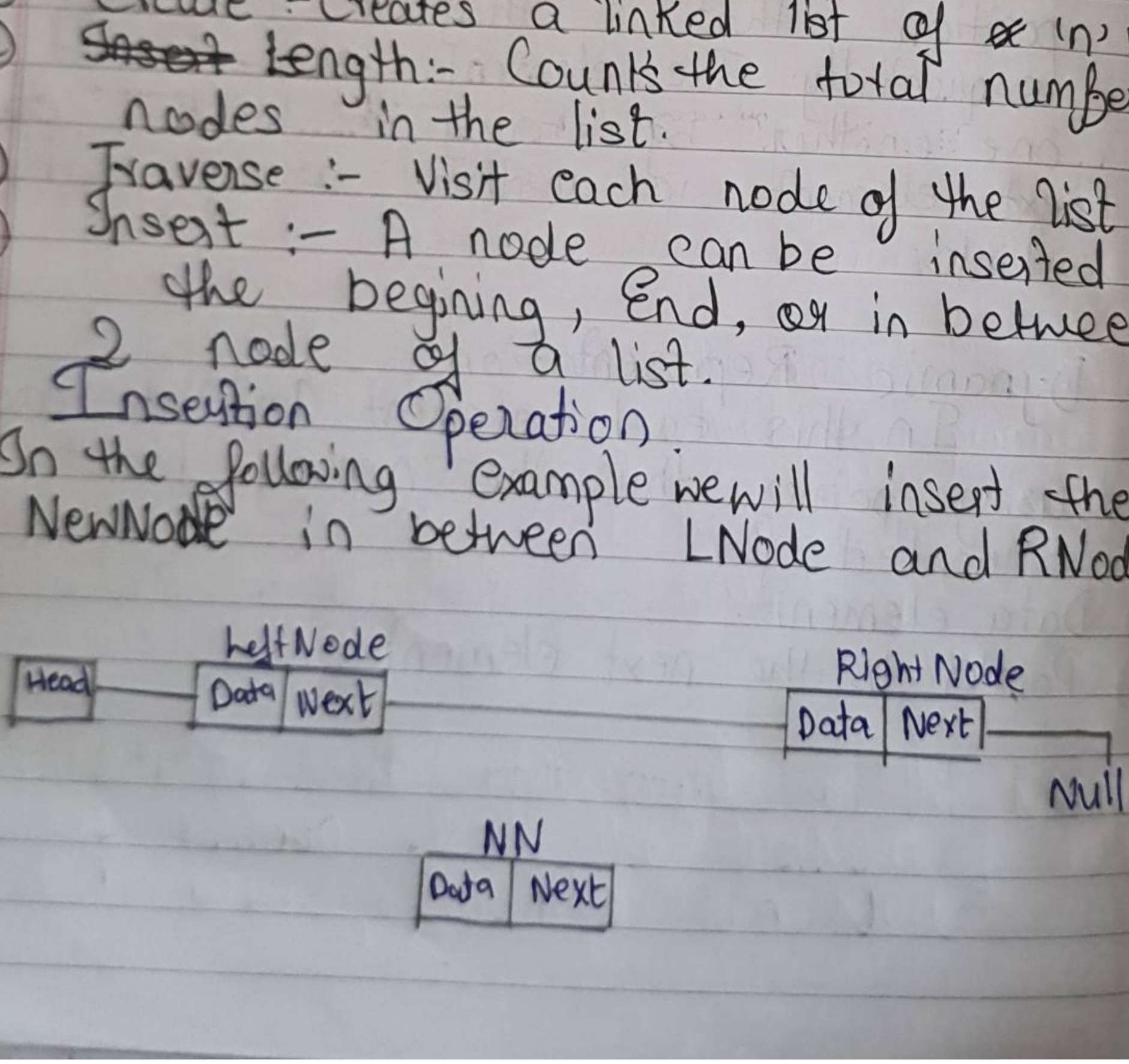
We can perform various operation on Linked list like

① Create :- Create's a linked list of 'n' Nodes
② ~~Insert~~ Length :- Counts the total number of nodes in the list.
③ Traverse :- Visit each node of the list
④ Insert :- A node can be inserted at the begining, End, or in between 2 node of a list.

* Insertion Operation

Q. In the following example we will insert the NewNode in between LNode and RNode.

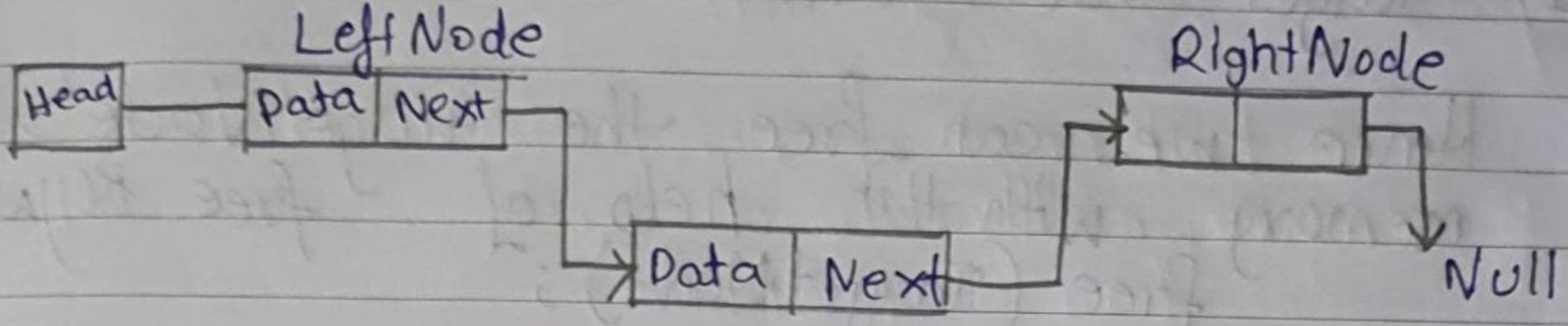
⇒



To perform this operation we can do it in 2 steps such that.

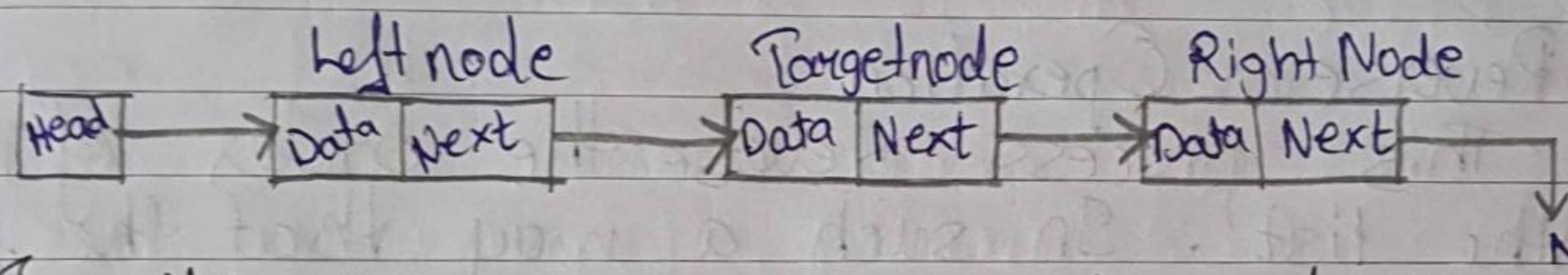
I) NewNode's Next will point to the RNode.
II) Next of the LNode will point to the NewNode

NewNode.next → RightNode;
LeftNode.next → NewNode;



* Deletion Operation.

Deletion from a linked list may be done either position wise or element wise.

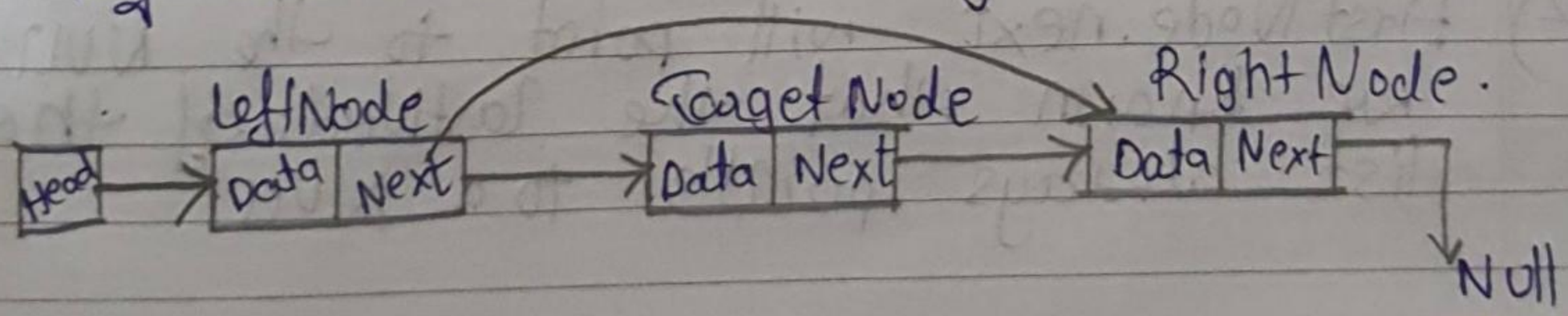


In this Example the node which we want to delete is consider as Target Node.

① To perform the deletion Operation we consider previous of the target Node as LeftNode.

② The LeftNode of TargetNode should point to the Next node of Target Node

I) LeftNode.next → TargetNode.next;



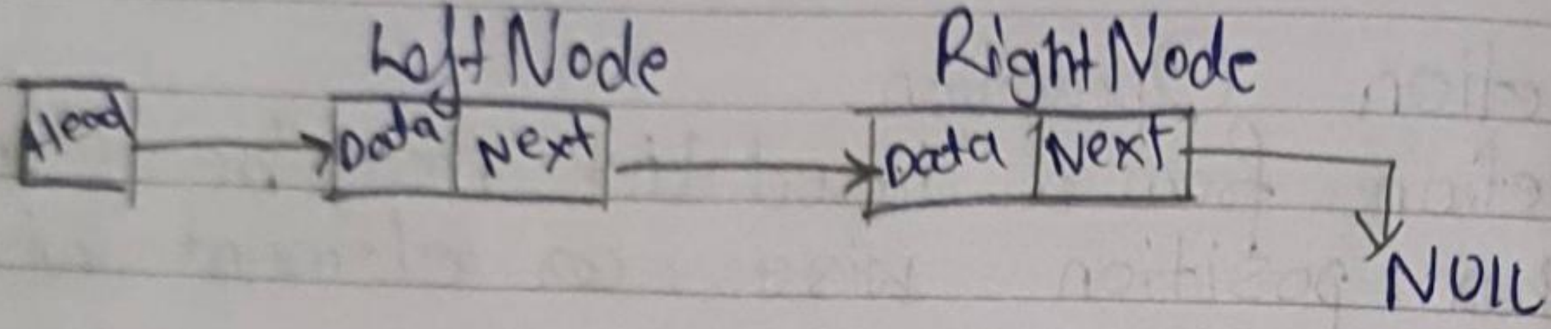
This will remove the linked between the LeftNode and the TargetNode

① To remove the pointing between Target Node and RightNode we need to peform

II) Target Node. next —> NULL;

Hence we can free the TargetNode's memory with the help of free kelyword

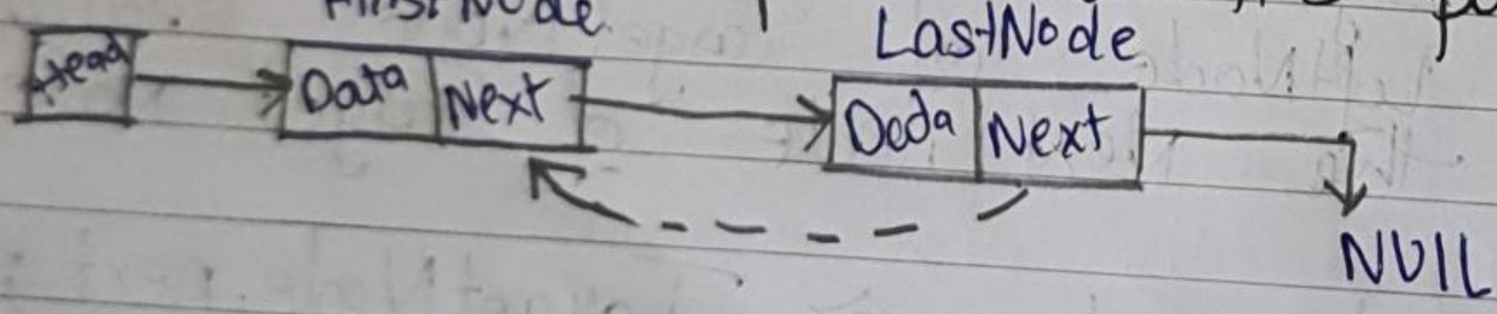
free (TargetNode);



* Reverse Operation.

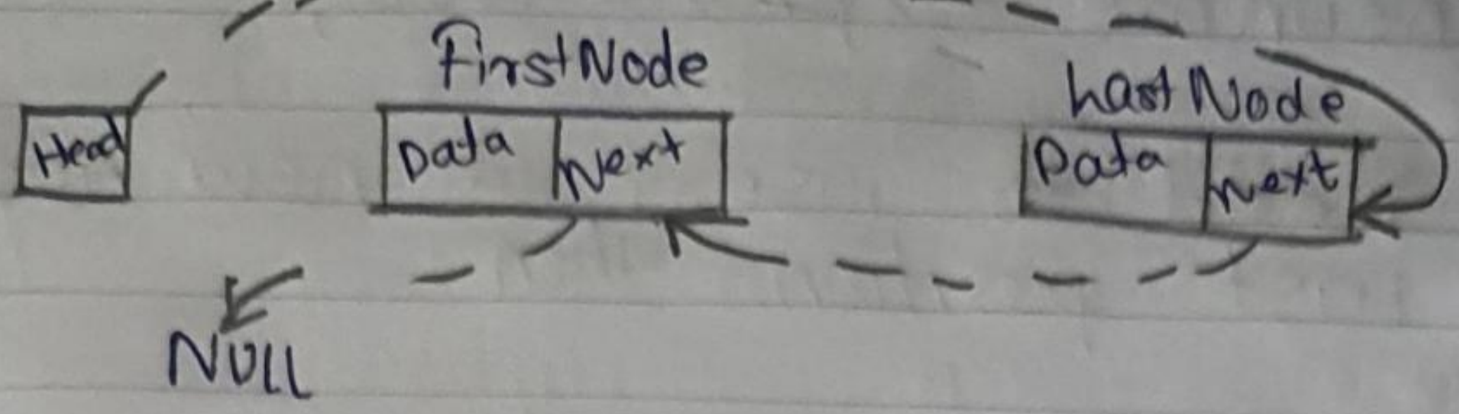
This process reverse the order of Node in the list. In such a way that the first Node becomes the last and wise versa.

1) In this scenario we will make sure that the last Node next which is pointing to NULL should point to it's previous Node



2) firstNode.next will point to the NULL. As we known that the lastNode of the linked list always point's to a. NULL

Also We'll make the head Node point to the Last Node to make it as a firstNode.



* Doubly Linked list.

In a ~~Sin~~ Singly Linked list each node contains only one pointer to the next node. Hence, traverse in only one direction.

In Doubly linked list each node contains 2 links one pointing to the previous node and other pointing to the next node.

In the given example below we are trying to insert a new node in an empty Linked list

Structure :-

```
struct node
{
    int data
    struct node *next,
                *prev;
};
```

head = NULL
head => newnode, newnode -> data = data.
newnode -> next = NULL,
newnode -> prev = NULL,

* If we need to insert a new node at the last position. then

tail.next = newnode.
newnode prev = tail
tail = newnode.

* Generalized Linked list
Generalized list G is finite sequence of n>=0 elements like a1, a2, an where ai is a list.
A Generalized linked list contain set of nodes. each node consist of coefficiently, Exponently, linkly feild

4/04/22

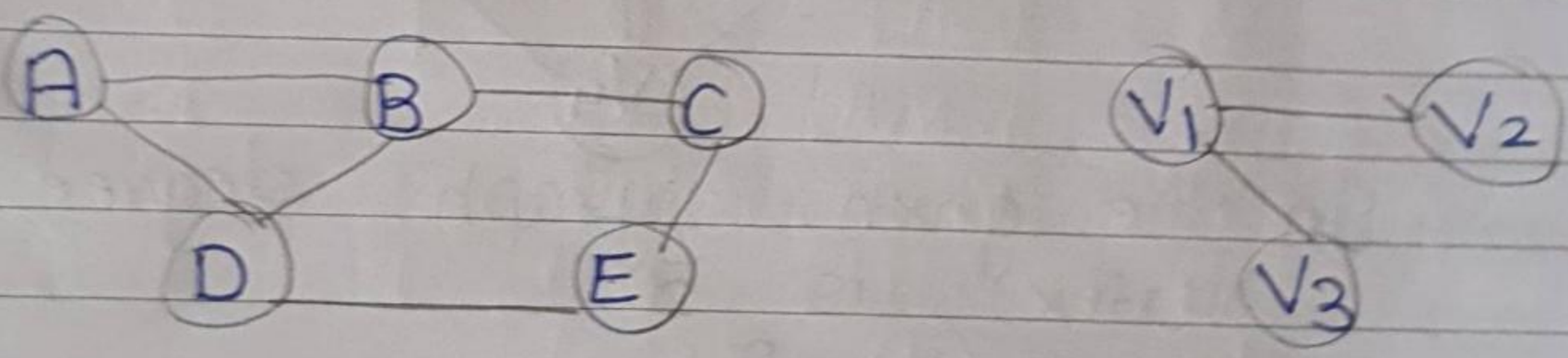
CHAPTER NO : 03. INTRODUCTION TO GRAPH

* Graph.

A Graph (G) is a collection of 2 Sets V and E where V is a finite nonempty set of vertices that is V of G and E is Non-empty finite set of edges that is E(G) connecting a pair of vertices.
Hence, we can represent a graph as G=(V,E)

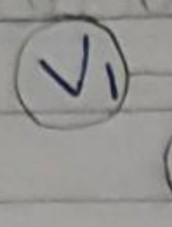
* Types of Graph

1. Undirected Graph : A Graph is undirected Graph if the pairs of vertices that make up the edges are unordered.



In the given graph an edge V_1, V_2 is same as V_2, V_1. A graph with undirected edges is known as Undirected Graph.

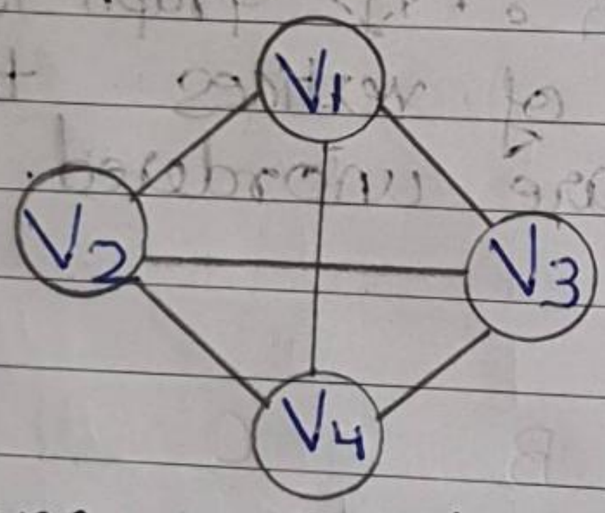
Directed Graph : In this each edges is represented by pair of ordered vertices. That edges have specific direction.



In the given graph V_1, V_2 is different than that of V_2, V_1 we can specify V_1, V_2 as we have a directed edge present between them.

* Degree Vertices

A Degree of vertices in an undirected graph is the total no of edges that vertex is connected to

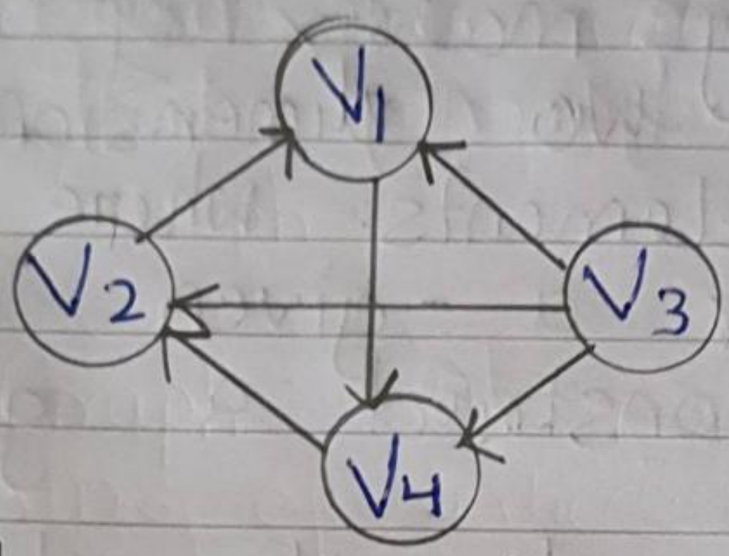


In the given graph degree of each vertex is 3.

degree $(V_1) = 3$
degree $(V_2) = 3$
degree $(V_3) = 3$
degree $(V_4) = 3$

* Degree of directed Graph:

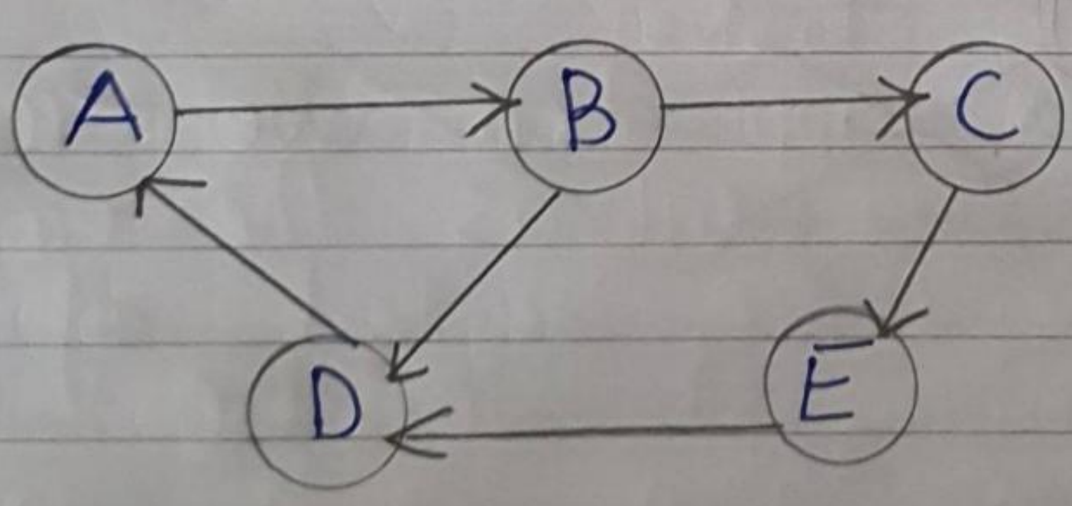
1) Indegree
If the graph is directed the indegree of a vertex is the no. of edges coming towards it.
A node or vertex whose indegree is 0 is called as a source node or vertex.



	In	Out
V1	2	1
V2	2	1
V3	0	2

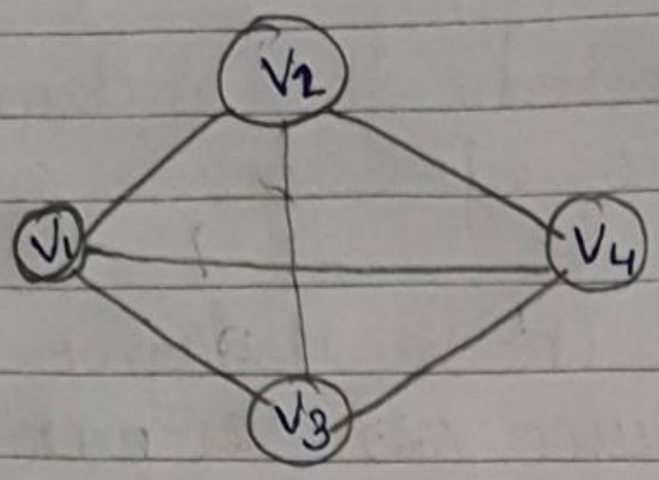
In the above graph V3 is a source node because it's indegree is 0.

2) Outdegree.
If (G) is a directed graph the outdegree of a vertex is no. of edges going out of it. a node whose outdegree is '0' is called a sink node.



Outdegree (B) = 2

* Graph Representation in Adjacency matrix

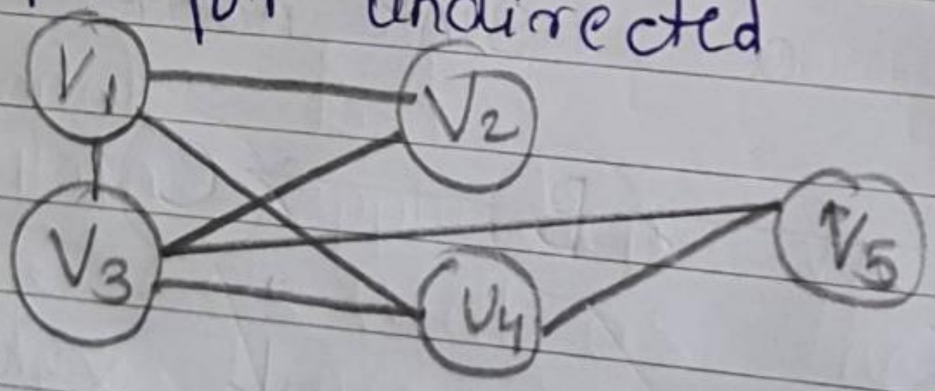


	V1	V2	V3	V4
V1	0	1	1	1
V2	1	0	1	1
V3	1	1	0	1
V4	1	1	1	0

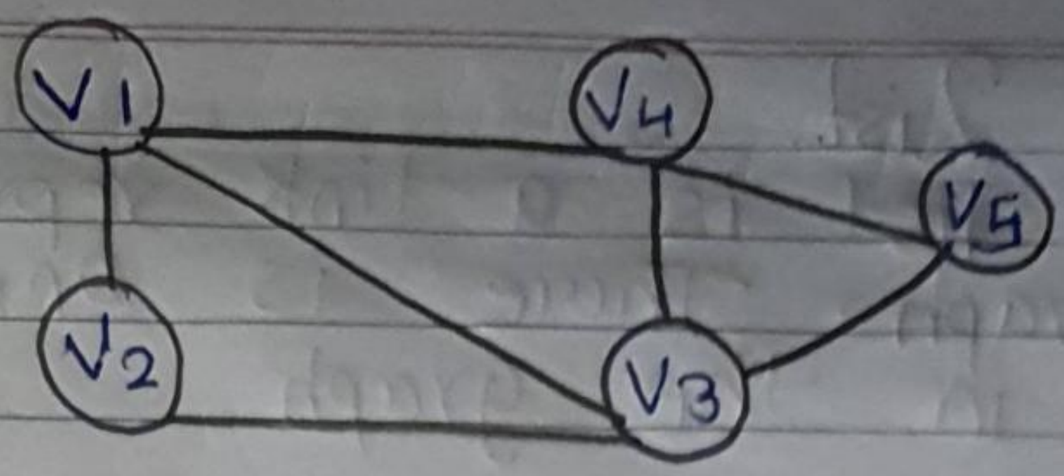
The Adjacency matrix of (A') of a Graph (G) is a two dimensional array of n×n elements. where n is no. of vertices (for a given adjacency matrix construct a graph)

	V1	V2	V3	V4	V5
V1	0	1	1	1	0
V2	1	0	1	0	0
V3	1	1	0	1	1
V4	1	0	1	0	1
V5	0	0	1	1	0

Graph for undirected

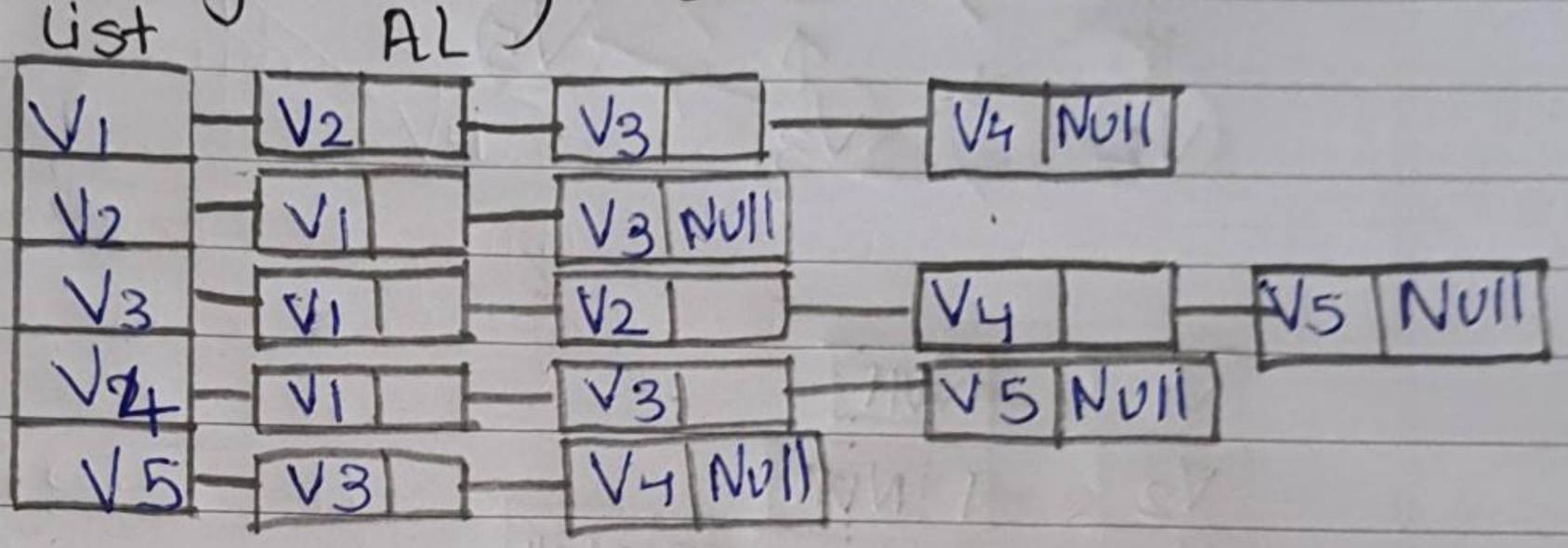


* Undirected

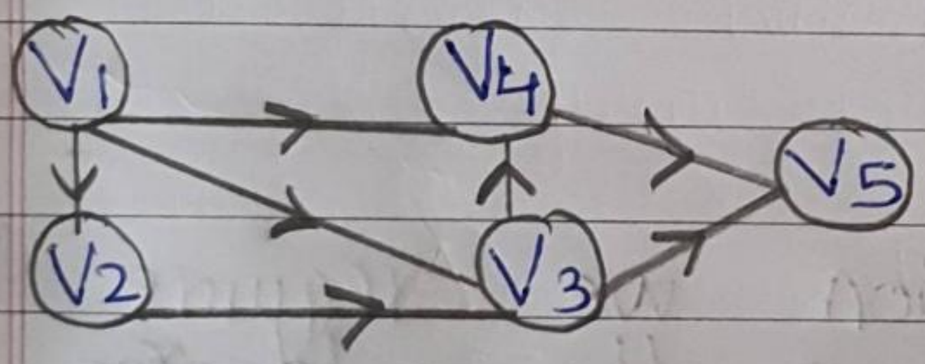


	V1	V2	V3	V4	V5
V1	0	1	1	1	0
V2	1	0	1	0	0
V3	1	0	0	1	1
V4	1	0	1	0	1
V5	0	0	1	1	0

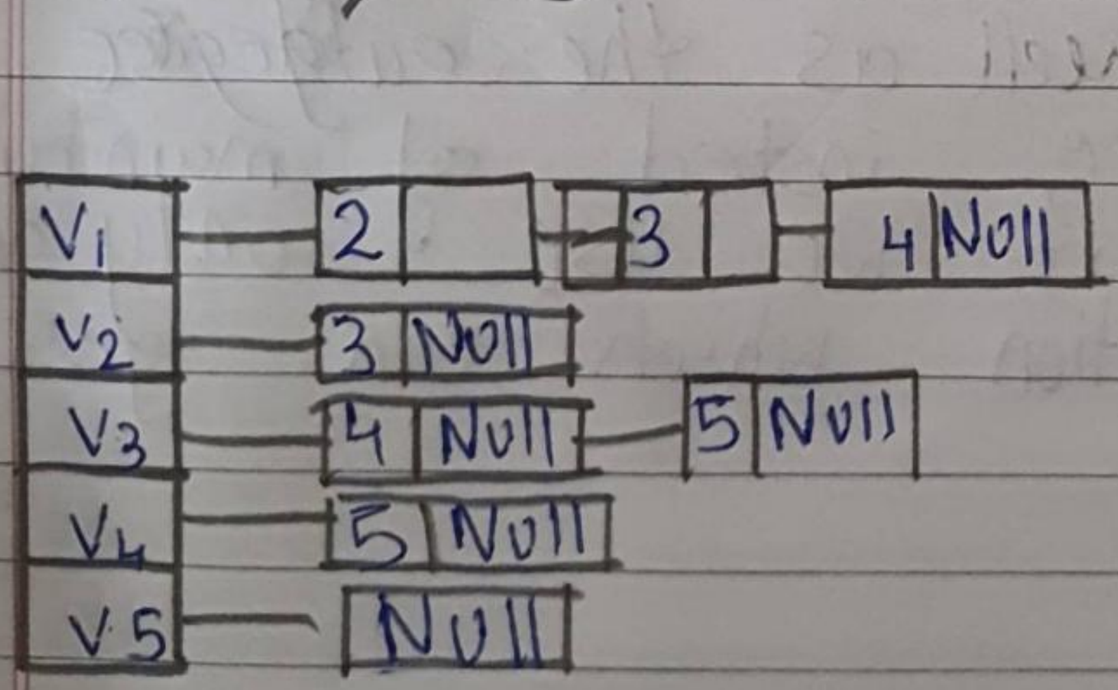
* Undirected. Adjacency list.



* Directed.

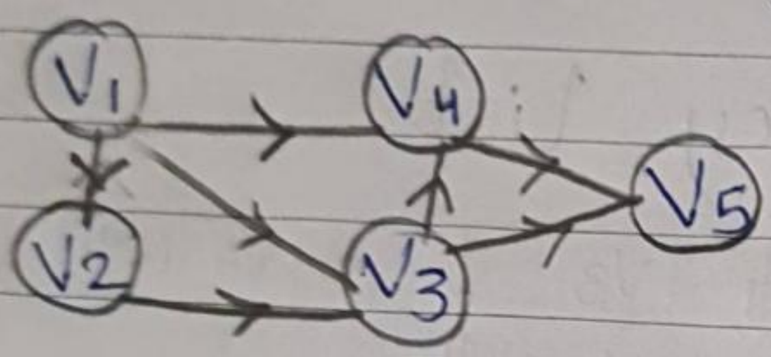


	V1	V2	V3	V4	V5
V1	0	1	1	1	0
V2	0	0	1	0	0
V3	0	0	0	1	1
V4	0	0	0	0	1
V5	0	0	0	0	0



* Adjacency List.
Adjacency list is a link representation of a graph. There is one for each vertex in the graph

* Inverse Adjacency List.
For every adjacency list we give outdegree of any vertex. To obtain the indegree we make use of inverse adjacency list



IMP * Orthogonal List
In certain application we require the indegree as well as the outdegree of a vertex. Hence, insted of maintaing 2 separted list we use only one graph representation which can give us both.

BFS - Queue - First come first served (Buffet)
DFS - Stack - Last come first served (Book)

* Graph Traversal