C.P. PATEL & F.H. SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA (ITM) & PGDCA PROGRAMME
PGDCA SEM I
PS01CDCA22: C AND DATA STRUCTURE
UNIT 4 Data Structures

Introduction to Data Structures

Data Structure:
A *data structure* can be defined as an arrangement of data in a computer's memory or even disk storage

Applications and advantages of data structures

The applications of data structure are:
- Compiler design
- Operating system
- Database Management System
- Statistical analysis package
- Numerical analysis package
- Computer graphics
- Artificial Intelligence
- Simulation
- Network analysis

Primitive and non-primitive data structures and operations on them

- Primitive Data Structures:
  - If the data contains a single value and
  - This value can be organized using primitive data types, the data structure is known as primitive data structure.
- Non-Primitive Data Structures:
  - If the data contains set of values and
  - They can be represented using non-primitive data types, the data structure is known as non-primitive data structures.
  - This non-primitive data structures can be classified as:
    - Linear Data Structures
    - Non-Linear Data Structures

Primitive data type	Non-primitive data type
1. A data type that is directly operated by machine level instruction is known as primitive data type.	1. A data type that is structured set of primitive data types is known as Non-primitive data type.
2. E.g. Integer, real, character, logical and pointer.	2. E.g. Array, stack, queue, linked list and file.
3. It is a fundamental data type.	3. It is a set of primitive data type.
4. There is no other name.	4. It is also known as composite or complex data type.

- **Linear and non-linear data structures**

Linear Data Structures:

- In linear Data structure the elements are stored in sequential order. The linear data structures are
  - **Array:** Array is a collection of data of same data type stored in consecutive memory location and is referred by common name
  - **Linked list:** Linked list is a collection of data of same data type but the data items need not be stored in consecutive memory locations.
  - **Stack:** A stack is a Last-In-First-Out linear data structure in which insertion and deletion takes place at only one end called the top of the stack.
  - **Queue:** A Queue is a First in First-Out Linear data structure in which insertions takes place one end called the rear and the deletions takes place at one end called the Front.

Non-Linear Data Structures:

- Non-Linear data structures store the elements on the bases of the hierarchical relationship among the data.
- The following are some of the Non-Linear data structure
  - **Trees:** Trees are used to represent data that has some hierarchical relationship among the data elements.
  - **Graph:** Graph is used to represent data that has relationship between pair of elements not necessarily hierarchical in nature. For example electrical and communication networks, airline routes, flow chart, graphs for planning projects.

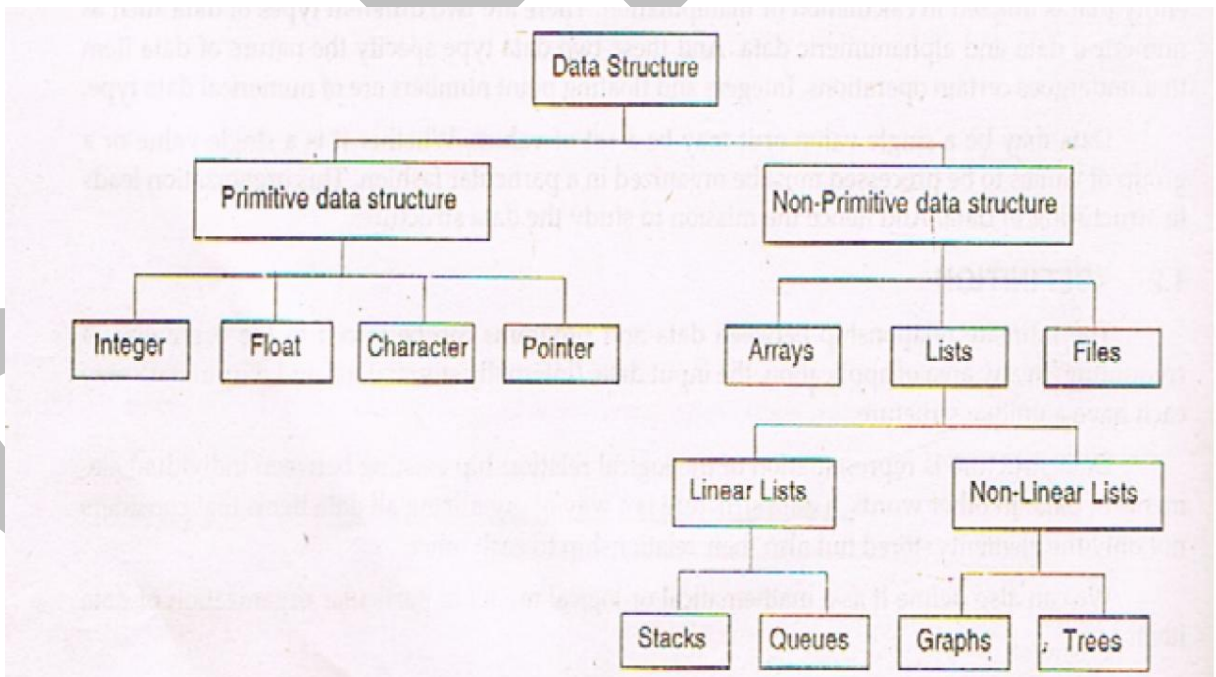


Fig. 1.1. Classification of Data Structure.

Operations on Data structure:

There are four basic operation carried out on primitive data structure. They are

1. CREATION
2. DESTROY
3. SELECTION
4. UPDATE

1. CREATION:

- An operation that is used to create a data structure is known as creation operation.
- A data structure can be created in PASCAL, FORTRAN, ALGO, PL\I plus and many other languages by suing a declaration statement.

2. DESTROY

- An operation that is provides the complementary effect of a CREATION operation is known as DESTROY operation.
- DESTROY means to delete the data structure.

3. SELECTION

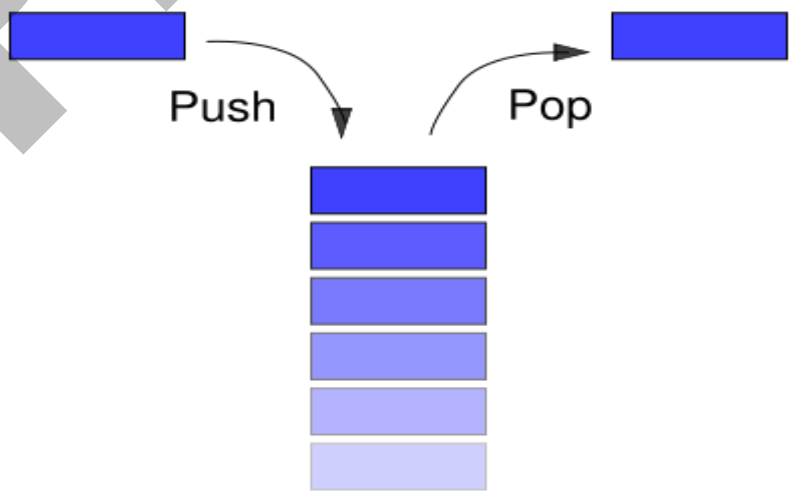
- An operation that is used to access data within the data structure is known as SELECTION operation.
- The form of selection operation is depends on the type of data structure being accessed.

4. UPDATE

- An operation that is used to change data in the data structure is known as UPDATE operation.
- E.g. An assignment statement (=) is a good example of an update operation.

Introductions to stacks, operations on stacks Algorithm: push, pop, peep and change

Definition: It is a data structure in which insertion and deletion of elements occur only at one end.



Characteristics:
A stack data structure has following characteristics:

- It is non-primitive data structure.
- Its nature is LIFO (Last In First Out).
- The insertion and deletion operation occur only at one end.
- It is linear data structure of variable size.
- The elements are removed from the opposite order from that in which they are added to the stack.

Operations on Stack are as given below:

- Push
- Pop
- Peep
- Change

Push:

- An insertion operation is known as Push.
- To insert an element on a stack.

Pop:

- A deletion operation is known as Pop.
- To delete an element from a stack.

Peep:

- To give the value of i^{th} element from a stack.

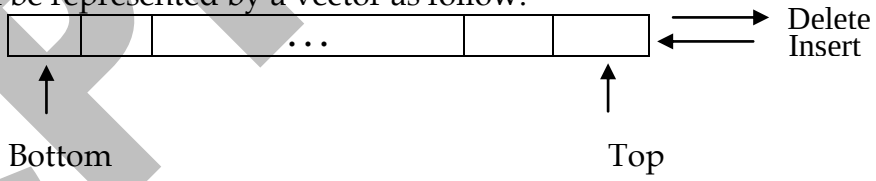
Change:

- To change the value of i^{th} element from a stack.

Top and Bottom of a stack:

- The most accessible element of a stack is known as Top of a stack.
- The least accessible element of a stack is known as Bottom of a stack.

Representation of a stack:
A stack can be represented by a vector as follow:



Algorithm:

[1]. An algorithm to insert an element into a stack.

PUSH (S, TOP, N, X) : This algorithm is to insert an element X into a stack S which is consisting of N elements. Top indicates the position of the top element from the stack.

Step -1 : [Check for Overflow **]**

```
If ( TOP >= N )
Then
        Write ('STACK OVERFLOW')
        Return
```

Step -2 : [Increment Top **]**
TOP ⟵ TOP + 1
Step -3 : [Insert an Element **]**
S[TOP] ⟵ X
Step -4 : [Finished **]**
Return

[2]. An algorithm to delete an element from a stack.

POP (S, TOP) : This algorithm is to delete an element from a stack S which is consisting of N elements. Top indicates the position of the top element from the stack.
Step -1 : [Check for Underflow **]**
If (TOP = 0)
Then
Write ('STACK UNDERFLOW')
Return
Step -2 : [Decrement Top **]**
TOP ⟵ TOP - 1
Step -3 : [Return an Element **]**
Return (S[TOP + 1])

[3]. An algorithm to give the value of i^{th} element from a stack.

PEEP (S, TOP, N, I) : This algorithm is to give the value of i^{th} element from a stack S which is consisting of N elements. Top indicates the position of the top element from the stack.
Step -1 : [Check for Underflow **]**
If ((TOP – I + 1) <= 0)
Then
Write ('STACK UNDERFLOW' ON PEEP)
Return
Step -2 : [Return an Element **]**
Return (S[TOP - I + 1])

[4]. An algorithm to change the value of i^{th} element from a stack.

CHANGE (S, TOP, N, I, X) : This algorithm is to change the value of i^{th} element to a new value X. Top indicates the position of the top element from the stack.

Step -1 : [Check for Underflow **]**
If ((TOP – I + 1) <= 0)
Then
Write ('STACK UNDERFLOW' ON CHANGE)
Return
Step -2 : [Change an Element **]**
S[TOP – I + 1] ⟵ X
Step -3 : [Finished **]**
Return

Applications of Stack :

1. Recursion
2. Polish Expression and their compilation
3. Stack Machine

Queues and their uses (only example – simulation)

Definition: It is a data structure in which insertion of an element occurs at one end and deletion occur at other end.

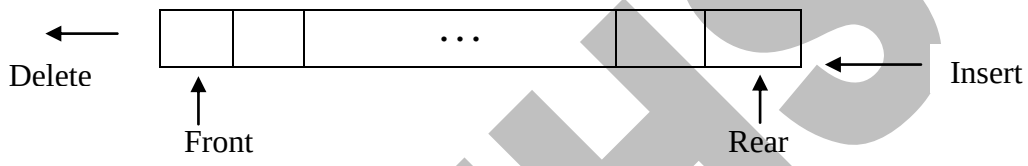
Characteristics:

A queue data structure has following characteristics:

- Its nature is FIFO (First in First Out).
- The insertion and deletion operation occur at opposite end.

Representation of a Queue:

A queue can be represented by a vector as follow:



Examples:

- People standing at Ticket Window
- Vehicles at a Traffic Signal

Types of a Queue:

1. Simple Queue **OR** Queue
2. Circular Queue
3. Deque (Double Ended Queue)
4. Priority Queue

Algorithm:

[1]. An algorithm to insert an element at the rear end of a Queue.

QINSERT (Q, F, R, N, X): This algorithm is to insert an element X at the rear end of a Queue which is consisting of N elements. F and R are the position of the front and rear element from the queue respectively.

Step -1: [Check for Overflow **]**

```
If ( R >= N )
Then
        Write ('QUEUE OVERFLOW')
        Return
```

Step -2 : [Increment rear pointer **]**

R ⟵ R + 1

Step -3 : [Insert an Element **]**

Q[R] ⟵ X

Step -4 : [Set Front pointer **]**

If (F = 0)

Thenp

F ⟵ 1

[2]. An algorithm to delete an element from the front end of a Queue.

QDELETE (Q, F, R, N): This algorithm is to delete an element from the front end of a Queue which is consisting of N elements. F and R are the position of the front and rear element from the queue respectively.

Step -1 : [Check for Underflow **]**

If (F = 0)

Then

Write ('QUEUE UNDERFLOW')

Return

Step -2 : [Delete Element **]**

X ⟵ Q [F]

Step -3 : [Check for Queue Empty **]**

If (F = R)

Then

F ⟵ R ⟵ 0

Else

F ⟵ F + 1

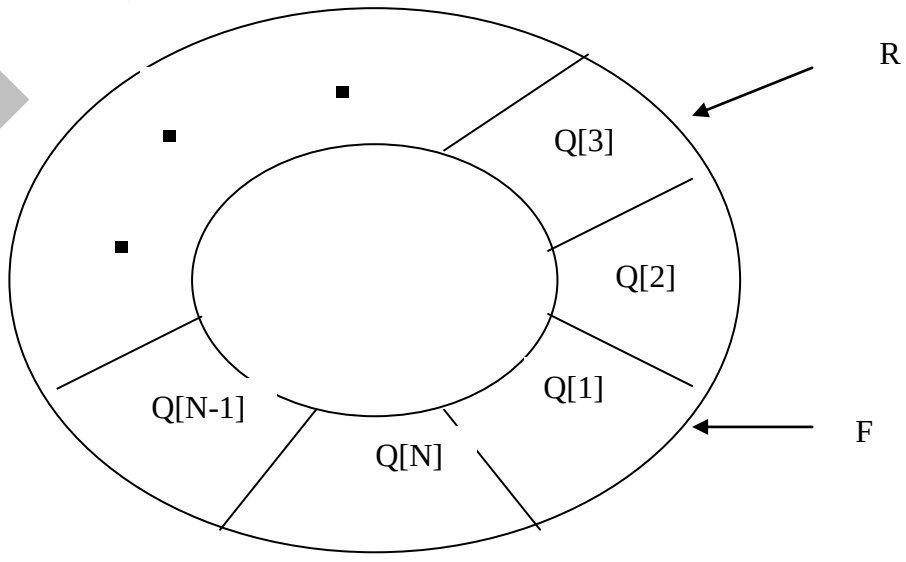
Step -4 : [Return an element **]**

Return (X)

Circular Queue

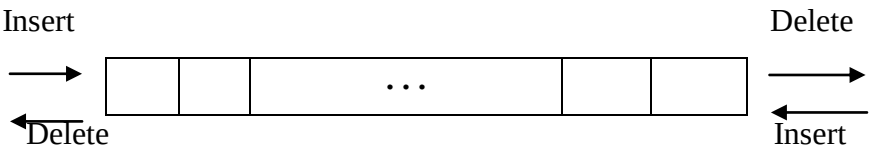
Definition: A queue in which the elements are arranged in the circular fashion is known as Circular queue.

Representation of a Circular Queue:



Double Ended Queue:

A queue in which insertion as well as deletion occurs at both the end is known as double ended queue.
It is also known as Deque.
A double ended queue can be represented by a vector as follow:



Types of Double Ended Queue:

1. Input restricted deque: The input restricted dequeue allows insertion at one end (it can be either front or rear) only.
2. Output restricted deque: The output restricted dequeue allows deletion at one end (it can be either front or rear) only.

Application of Queue:

A well known application of a queue is **Simulation**

Introduction to linked lists

A data structure that contains not only a data field but also contains pointer field is known as linked list.

A node of a linked list consist two fields:
- An information field is represented by INFO
- A pointer field is represented by LINK

A node structure is as follow:

NODE

INFO	LINK

- **Types of linked lists**
  - Singly linked list
  - Circular Linked list
  - Doubly linked list

Singly Linked List

A node structure of singly linked list is as follow.

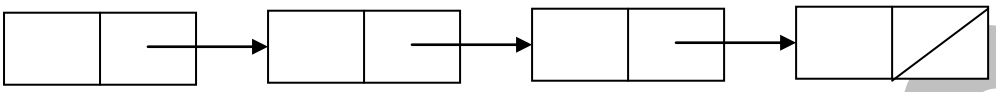
NODE

INFO	LINK

A node of the linked list consist two fields:

- An information field is represented by INFO
- A pointer field is represented by LINK. **It is pointing to the next node of a list.**

A linked list in which pointer fields of each node is pointing to the next node is known as singly linked list.



FIRST

In Singly linked list,

- An address of the first node is represented by **FIRST**.
- A right pointer field of last node is set to **NULL**.

Circular Linked List

A node structure of Circular linked list is as follow.

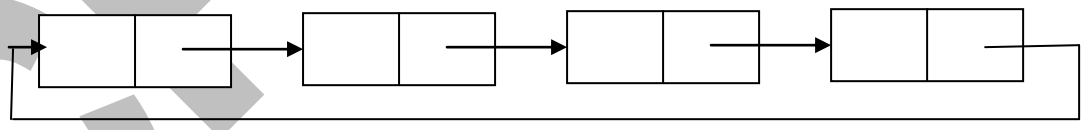
NODE

INFO	LINK

A node of the linked list consist two fields:

- An information field is represented by INFO
- A pointer field is represented by LINK.

A Singly linked list in which pointer fields of last node is pointing to the first node is known as circular linked list.



FIRST

In Circular linked list,

- An address of the first node is represented by **FIRST**.
- A right pointer field of last node is set to address of FIRST node.

Doubly Linked List

A node structure of doubly linked list is a follow.

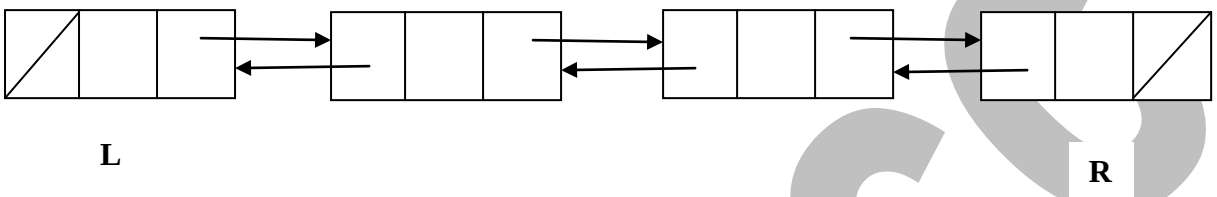
LPTR	INFO	RPTR

LPTR – pointing to the left node

RPTR – pointing to the right node

INFO – actual value

A data structure that has an information field and two pointers fields is known as doubly linked list. Each of the pointer field contains the address of left or right node. An information field contains the actual data about node.



In doubly linked list,

- An address of the first node is represented by **L**.
- An address of the last node is represented by **R**.
- A left pointer field of first node and right pointer field of last node is set to **NULL**.

Trees

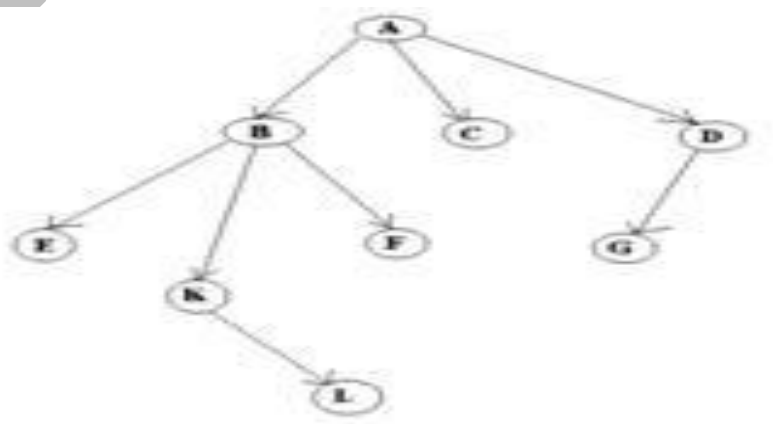
Introduction to Trees

Trees are very flexible, versatile and powerful data structures that can be used to represent data items possessing hierarchical relationship between the grandfather and his descendants and in turn their descendants and so on.

A Tree is a non-linear data structure in which items are arranged in a stored sequence. It is used to represent hierarchical relationship existing amongst several data items.
The graph theoretic definition of tree is :

It is a finite set of one or more data items (nodes) such that

1. **There is a special data item called the root of the tree.**
2. **And its remaining data items are partitioned into number of mutually exclusive (i.e, disjoint) subsets, each of which is itself a tree. And they are called Subtrees.**



Tree Terminology

Directed Tree :-

A directed tree is a directed graph $T=(V,A)$ with a designated node $r \in V$, the root, such that for each node $v \in V$, there is exactly one path from r to v in T.

Root:-

It is specially designed data item in a tree. It is the first in the hierarchical arrangement of data items. In the above tree, A is the root node.

Leaf :-

The node with degree 0 is a leaf or terminal node.

Node:-

Each data item in a tree is called a node. It is the basic structure in a tree. It specifies the data information and links (branches) to other data items. There are 9 nodes in the above tree.

Terminal Node (s):-

A node with degree zero is called a terminal node or a leaf. In the above tree, there are 4 terminal nodes. They are E, L, F, G.

Non-Terminal Node (s):-

Any node (except the root node) whose degree is not zero is called non-terminal node. Non-terminal nodes are the intermediate nodes in traversing the given tree from its root node to the terminal nodes (leaves). There are 3 non-terminal nodes.

Siblings :-

The children nodes of a given parent node are called siblings. They are also called brothers. In the above tree,

E, K, F are siblings of node B.

Level:-

The entire tree structure is leveled in such a way that the root ode is always at level 0. Then, its immediate children are at level 1 and their immediate children are at level 2 and so on upto the terminal nodes. In general, if a node is at level n, then its children will be at level n+1. In the 4 levels.

Edge:-

It is a connection line of two nodes. That is, the line drawn from one node to another node is called an edge.

Path:-

It is a sequence of consecutive edges from the source node to the destination node. In the above tree, the path between A and I is given by the node pairs,

(A,B), (B, K) and (K,L)

Depth:-

It is the maximum level of any node in a given tree. In the above tree, The root node A had the maximum level. That is the number of levels one can descend the tree from its root to the terminal nodes (leaves). The term height is also used to denote the depth.

Forest:-

It is the set of disjoint trees. In a given tree, if you remove its root node then it becomes a forest in the above tree, there is forest with three trees.

Application of Trees

1) Basically it is used for Searching.
2) One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

file system

```
————–
   /   <-- root
 /    \
...      home
    /       \
  ugrad      course
   /       /   |   \
 ...      cs101 cs112 cs113
```

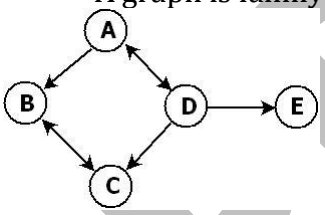
3) If we organize keys in form of a tree (with some ordering e.g., BST), we can search for a given key in moderate time (quicker than Linked List and slower than arrays). Self-balancing search trees like AVLand Red-Black trees guarantee an upper bound of O(Logn) for search.

4) We can insert/delete keys in moderate time (quicker than Arrays and slower than Unordered Linked Lists). Self-balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(Logn) for insertion/deletion.

5) Like Linked Lists and unlike Arrays, Pointer implementation of trees doesn't have an upper limit on number of nodes as nodes are linked using pointers.

Graph:-

A graph is family defined as a collection of a set of nodes and as set of edges.



Directed Edge:-

In a graph, an edge which is directed from one node to another is called Directed Edges.

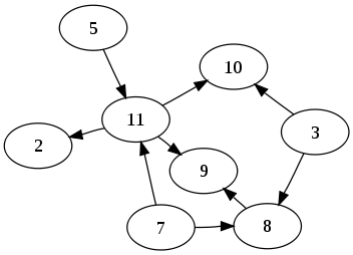
Undirected Edge:-

While an edges, which has not specific direction is called Undirected Edges.

Types of Graph:-

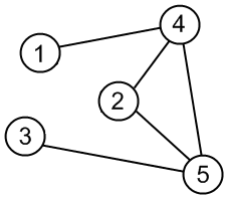
Directed Graph (Digraph):-

A graph in which every edges is directed is called Direct graph.



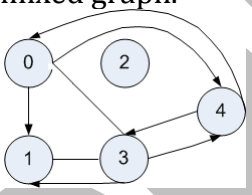
Undirected Graph:-

A graph in which every edge is undirected is called Undirected graph.



Mixed Graph:-

If some of edges are directed and some are undirected in a graph then the graph is a mixed graph.



Loop (Sling, Self Loop):-

An edge of a graph which joins a node to itself is called a Loop(Sling).