An Introduction to Data Structures

(Quadrant 1- Quadrant 4)

By

Dr. Parul Saxena
Convener and Head,
Department of Computer Science,
Soban Singh Jeena University,
Campus Almora, 263-601
Uttarakhand, India
E-mail:-parul_saxena@yahoo.com

Month & Year: May 2022

Quadrant 1

Data Structures

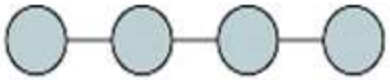
- Data structure :The logical arrangement of data elements, combined with the set of related operations.

- It may be represented by the triplet :

  <Data, Functions, Axioms>

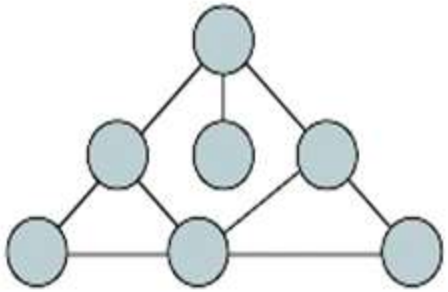
Linear and Nonlinear data structures

- Linear : Array, Stack and Queue etc.

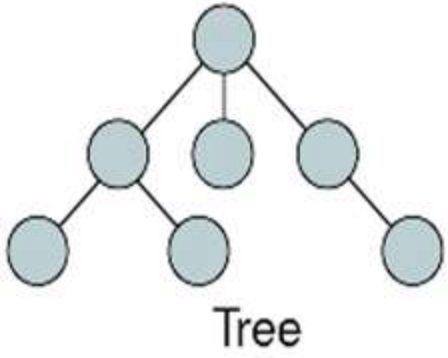


Linear list

- Non Linear: Graph, Tree etc.



Graph



Tree

Array

- An array is a sequenced collection of similar type of elements.
- The array elements can be referred by an index (0 to size-1).

10	20	30	40	50	60	70	80	90	100
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Array Operations

There are several operations that can be performed on an array:

Operation	Description
Traversal	Processing each element in the array
Searching	Finding the location of an element with a given value
Insertion	Adding a new element to an array
Deletion	Removing an element from an array
Sorting	Organizing the elements in some order
Reversing	Reversing the elements of an array

Traversal

```
/*To traverse the entire array*/
void  display(int A[])
{int  i, Max=10;
printf("\n");
for(i=0;i<Max;i++)
printf("%d\t", A[i]);}
```

Output:

```
10    20    30    40    50    60    70    80    90    100
```

Searching

```
/*To search the given element in the
  array*/
void search (int A[], int num)
{ for(int i=0;i<Max; i++)
  { if(A[i]== num)
      {printf("\n\n The element %d is
      present at position %d",num,i);
      return;}
  }
```

Contd…

```
if(i==Max)
printf("\n\n The element %d is not present
   in the array", num);
}
```

For example if we want to search num 30 then the output will be:

The element 30 is present at position 2.

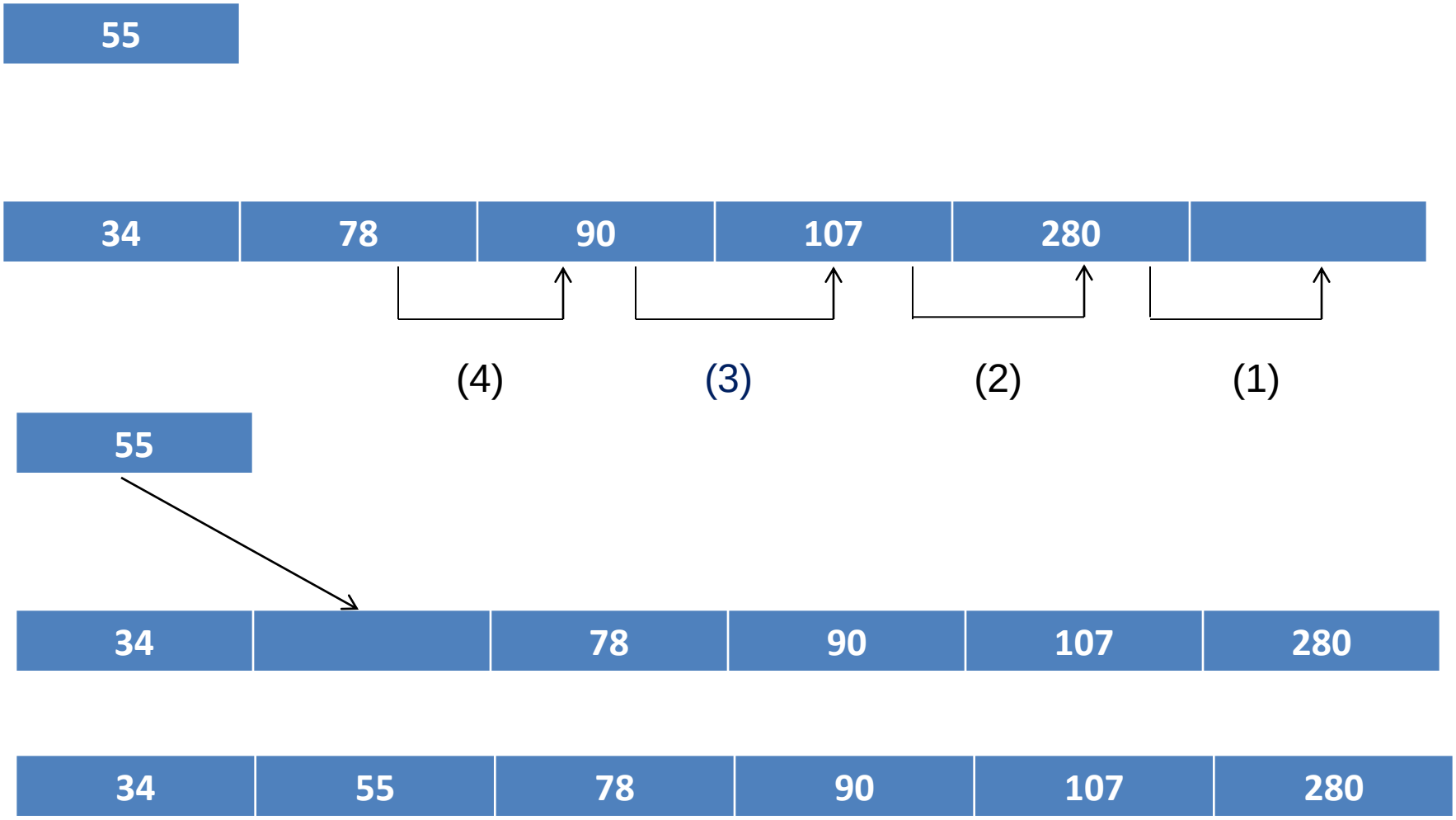
//as first element is at location 0

Quadrant 2

Insertion in Array

```
/*To insert an element num at given position
   pos*/
void insert(int A[],int pos, int num)
{ int i, Max=6;
for(i=Max-1;i>=pos;i--)
A[i]=A[i-1];
A[i]=num;
}
```

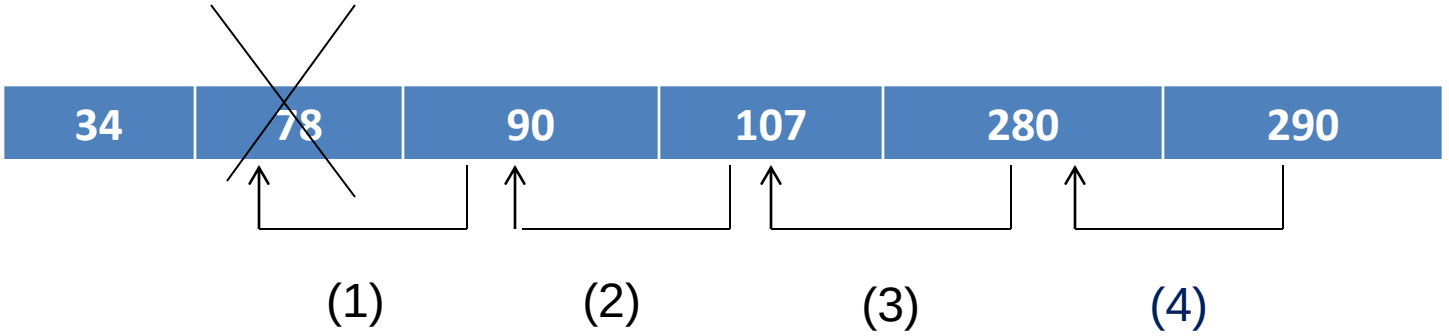
Insertion of Elements



Deletion in Array

```
/*To deletes an element from the given position
   pos*/
void delete(int A[],int pos)
{
int i, Max=6;
for(i=pos;i<Max;i++)
A[i-1]=arr[i];
A[i-1]=0;
}
```

Contd…



34	90	107	280	290	0

Sorting

```
/*To sorts an array in ascending order*/
void sort(int A[],int size)
{ int i, temp, j;
for(i=0;i<size;i++)
{for(j=i+1;j<size;j++)
{If(A[i]>A[j])
{temp=A[i];
A[i]=A[j];
A[j]=temp;}}}}
```

Sorting

30	50	20	10	40
i=0	j=1	j=2(swap)		
20	50	30	10	40
i=0			j=3(swap)	
10	50	30	20	40
i=0				j=4
10	30	50	20	40
	i=1	j=2	j=3(swap)	
10	20	50	30	40
	i=1			j=4
10	20	30	50	40
		i=2	j=3	j=4
10	20	30	50	40
			i=3	j=4(swap)
10	20	30	40	50

Reversing

```
/*Reversing the entire array*/
{int i;
for(i=0,j=Max-1;i<MAX/2;i++,j--)
   {    int temp =A[i];
        A[i]=A[j];
        A[j]=temp;}}
```

Input:

30	70	90	54	26	86	45

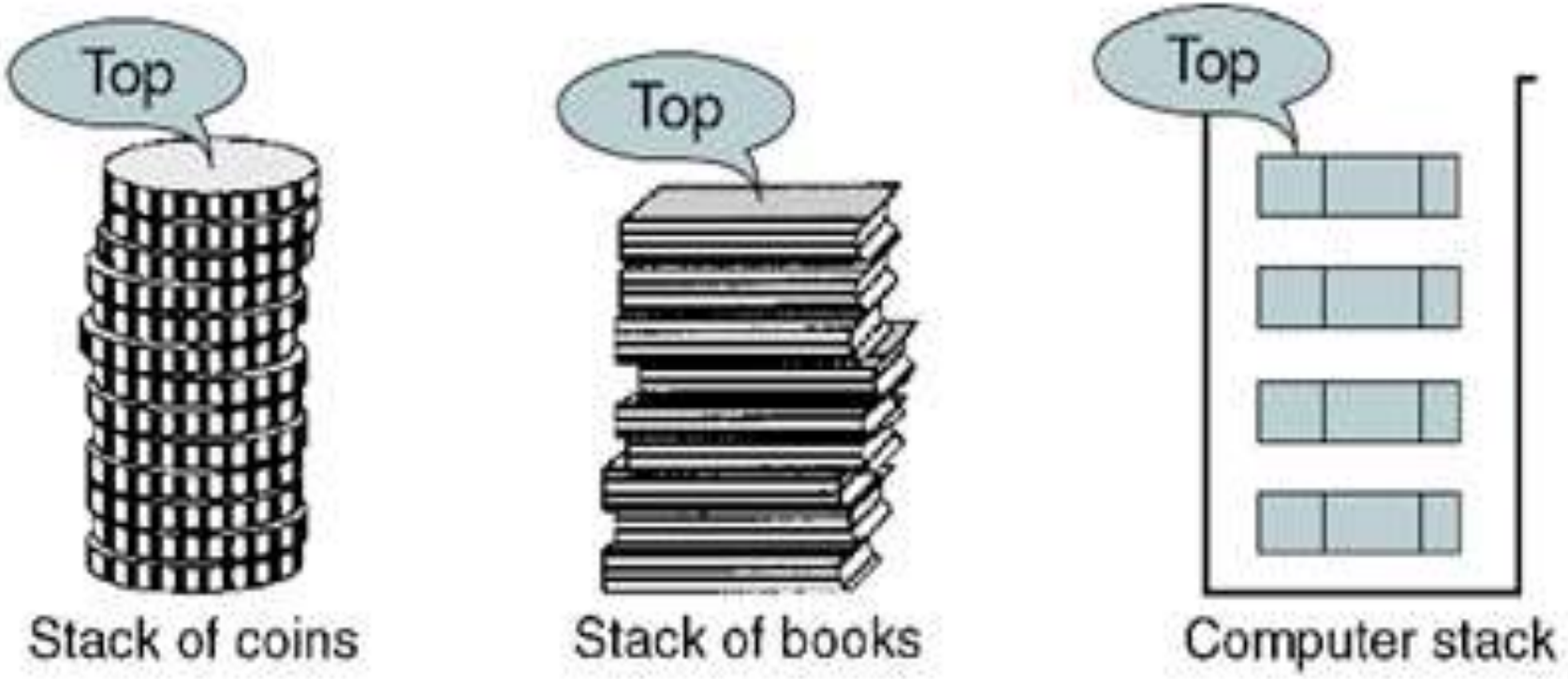
Output:

45	86	26	54	90	70	30

Quadrant 3

Stack

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.



Initializing the Stack

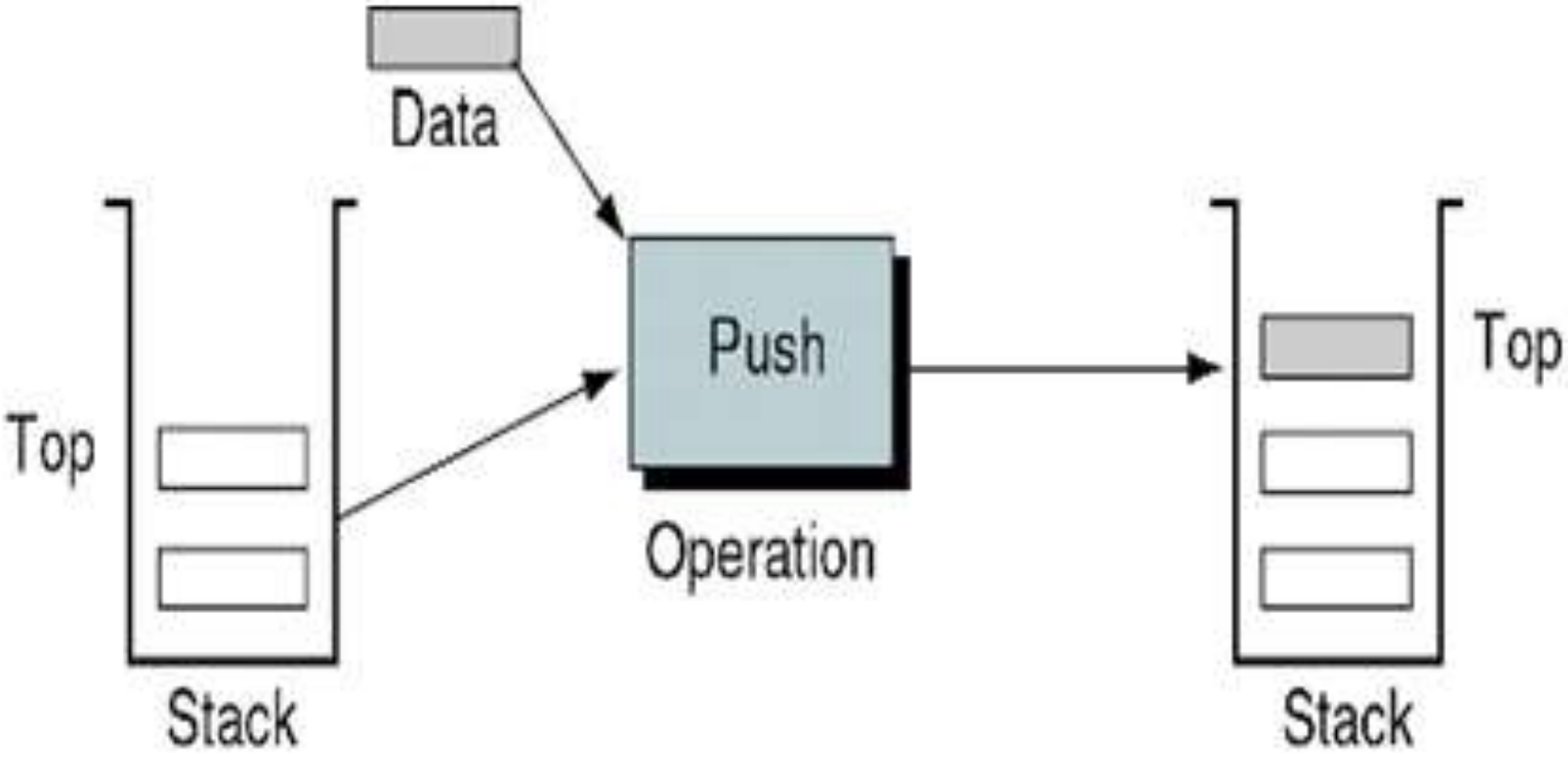
```
struct stack
{
int arr[MAX];
int top;
};
/*Initializes the stack*/
void init(struct Stack *S)
{ S->top=-1;}
```

top=-1

Push Operation

```
/*adds an element to the stack*/
void push(struct stack *S, int item)
{  if(S->top==MAX-1)
        {  printf("\n Stack is Full");
            return;}
   S->top=S->top+1;
   S->arr[S->top]=item;
}
```

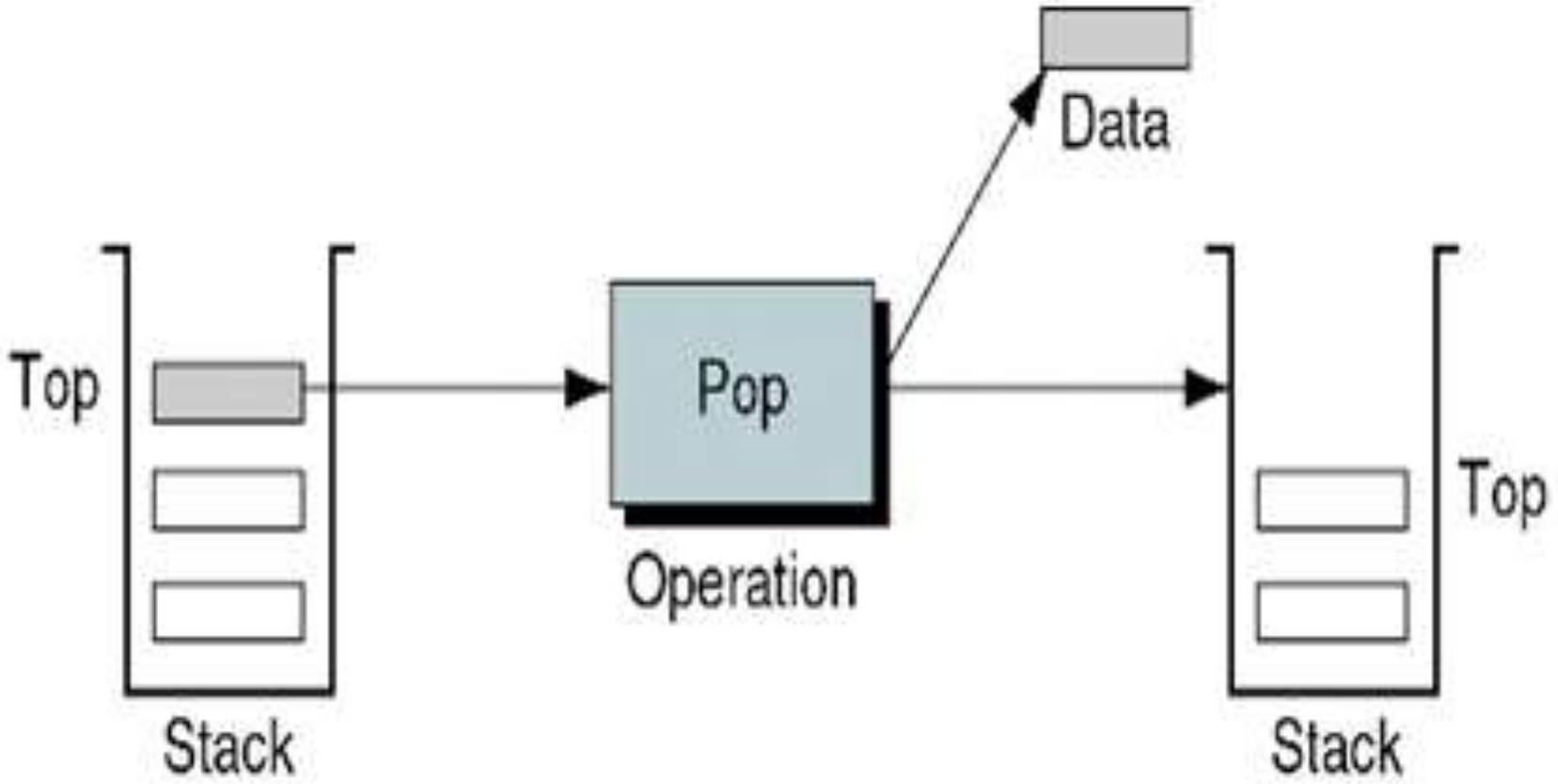
Push Operation



Pop Operation

```
int pop(struct stack *S)
{ int data;
If(S->top==-1)
{printf("\n Stack is empty."); return NULL;}
data=S->arr[S->top];
S->top=S->top-1;
return data;
}
```

Pop Operation



Queue

- A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle.
- Data can be inserted at one end (rear) and deleted from the other end (front).

A queue (line) of people

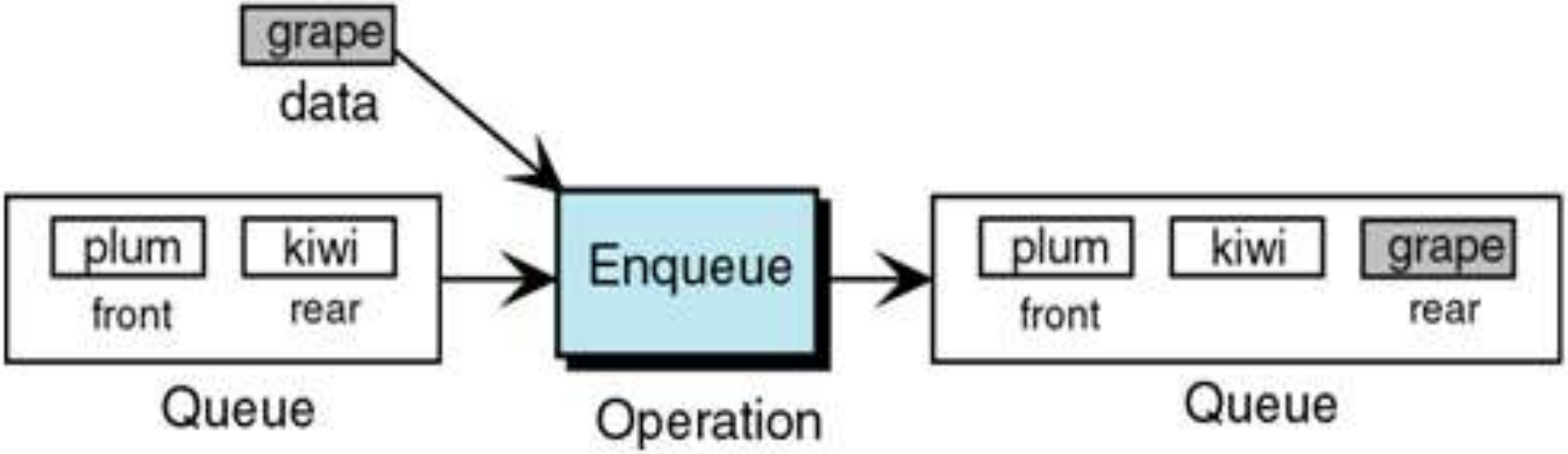
Initializing the Queue

```
struct  queue
{  int front, rear, a[MAX]; };
void init ( struct Queue  *Q )
{Q->front = Q->rear = - 1;}
```

A computer queue

Enqueue Operation

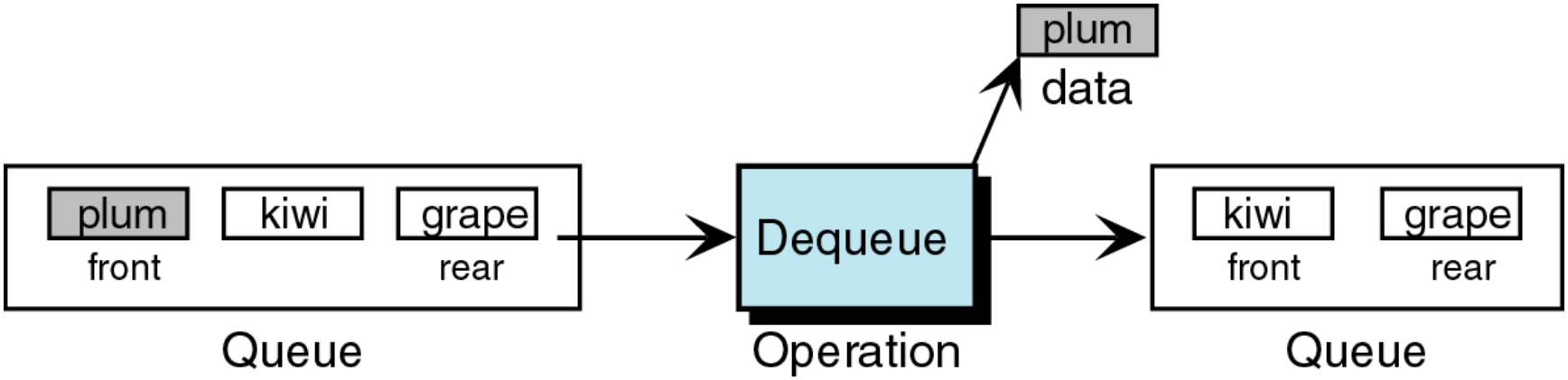
```
void enqueue (struct Queue *Q, int item)
{if ( Q->rear = = ( MAX - 1 ) )
printf("\nQUEUE IS OVERFLOW.....”;);
else  a[ + + rear ] = data; }
```



Dequeue operation

```
void dequeue (struct Queue *Q)
{
if( Q->front = = Q->rear )
printf("\n QUEUE IS UNDERFLOW.....");
else
printf("\n DELETED ELEMENT IN QUEUE IS
 %d",a[++front]);
}
```

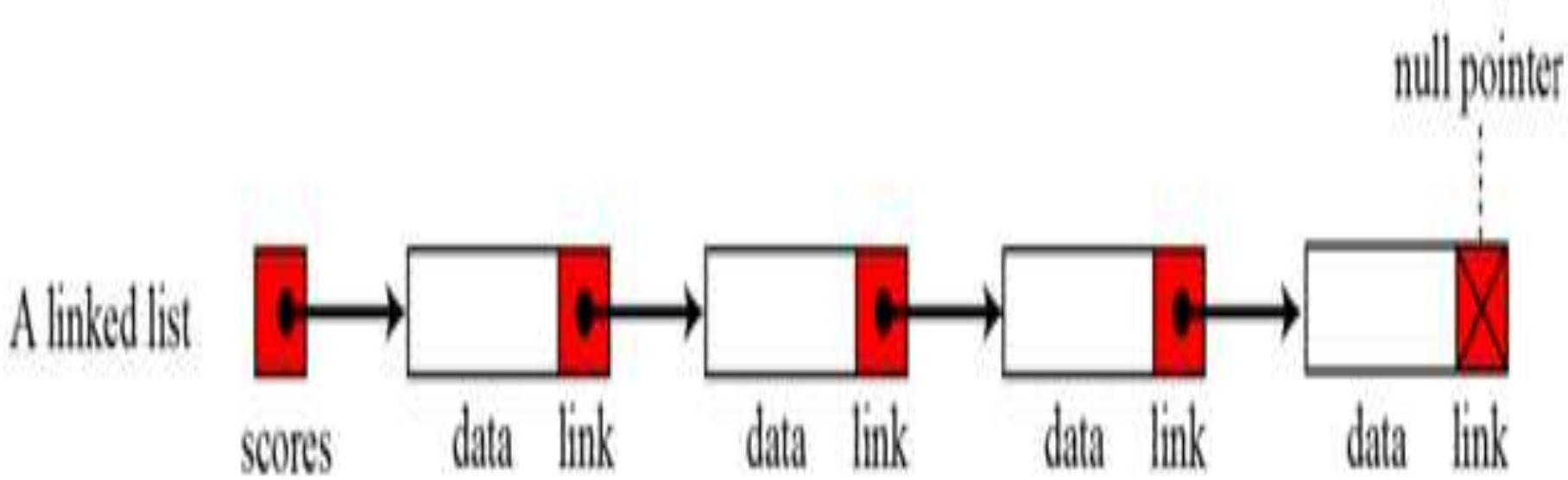
Dequeue Operation



Quadrant 4

Linked Lists

A linked list is a collection of data in which each element contains two parts: data and link to the next element.



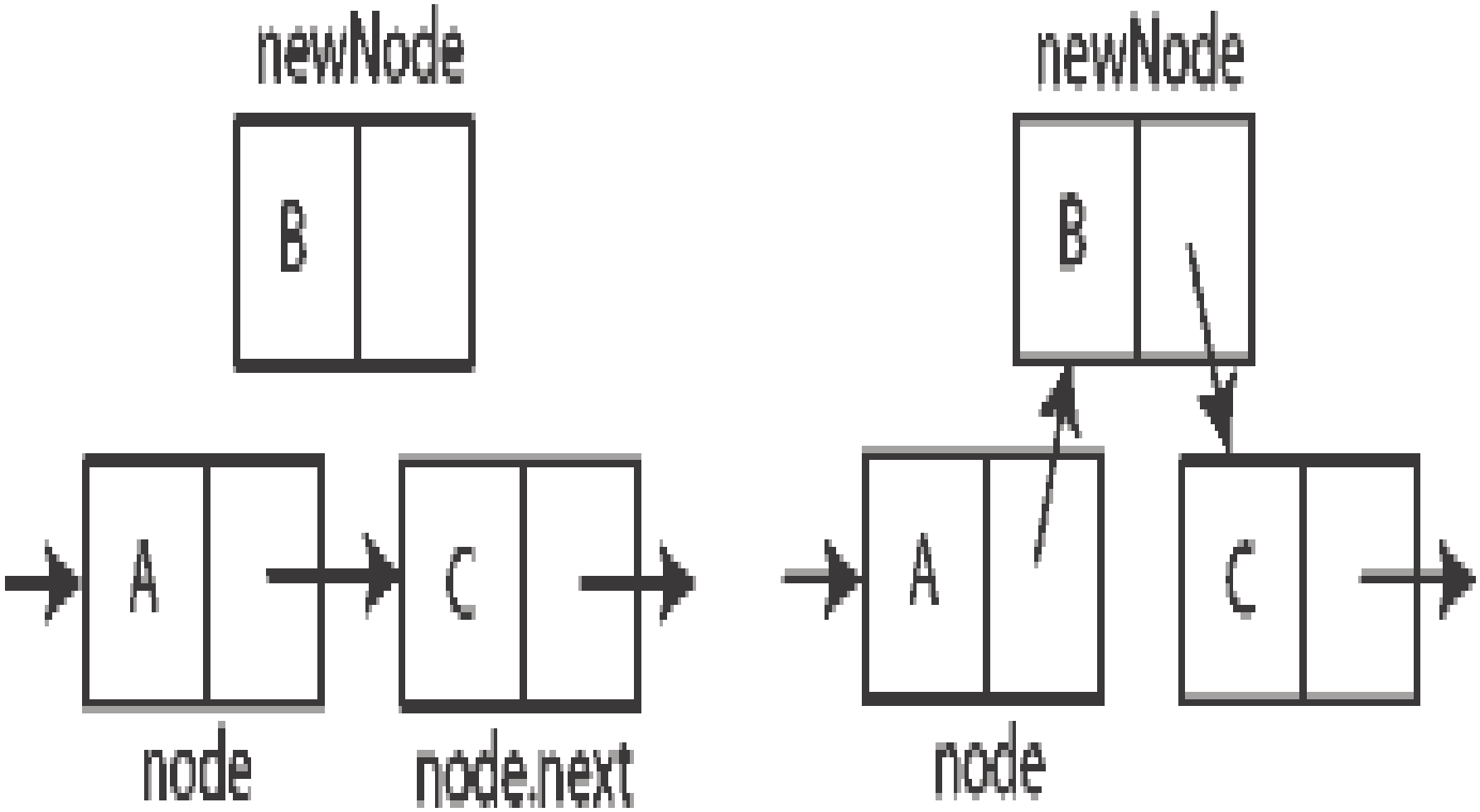
Creation of Linked List

```
struct Node{int  data;
Node *Link;};
Node * createNewNode(int X)
{ Node *p=new Node;
If (p!=NULL)
   {p->data=X;
   p->link=NULL; }
return p;}
```

Insertion of New Node

```
void Insert_after (node *newnode, node *p)
{If (p!=pTail)
   {
     newnode->next=p->next;
     p->next=newnode;
   }
else
   insert_Last (newnode);
```

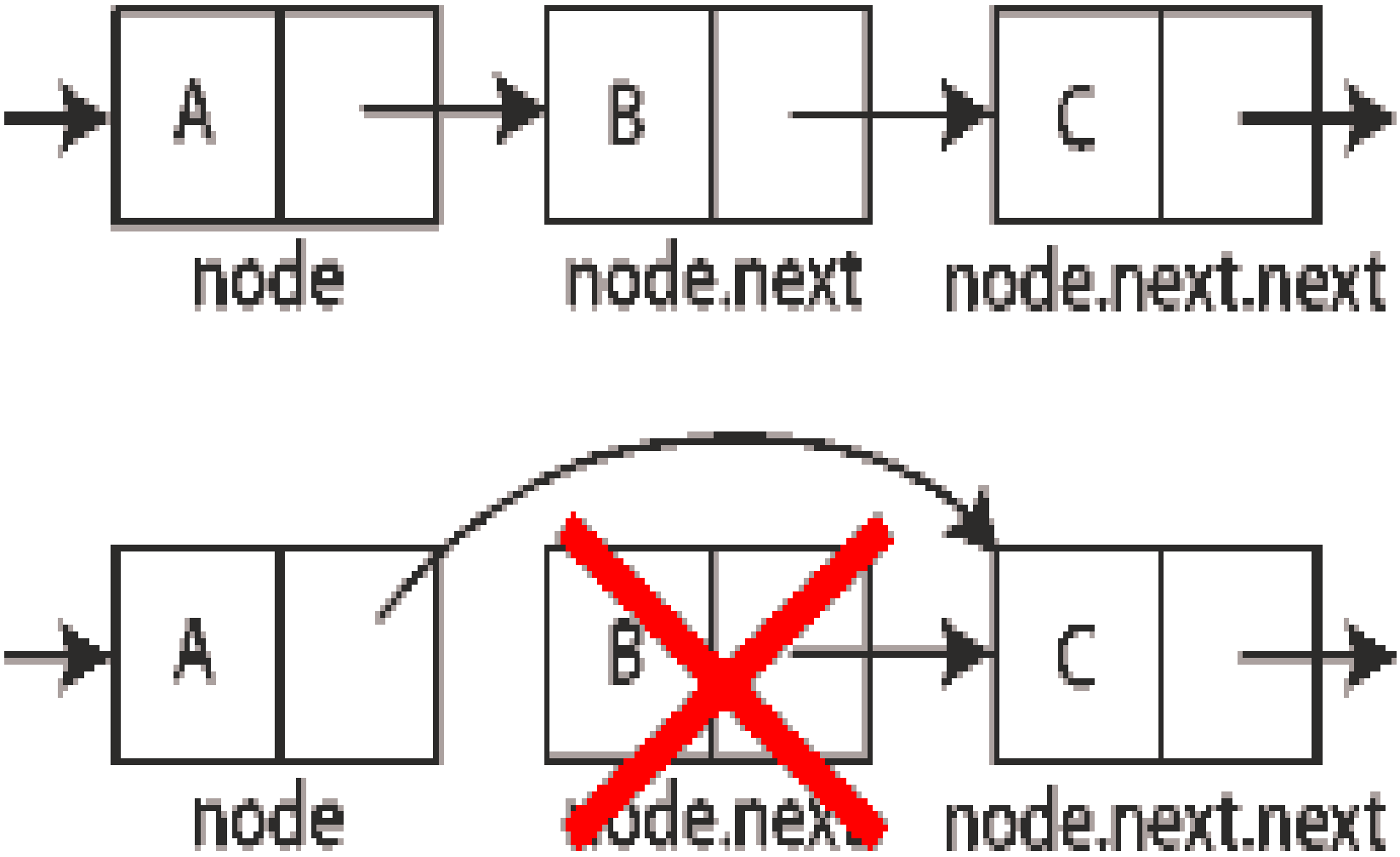
Insertion



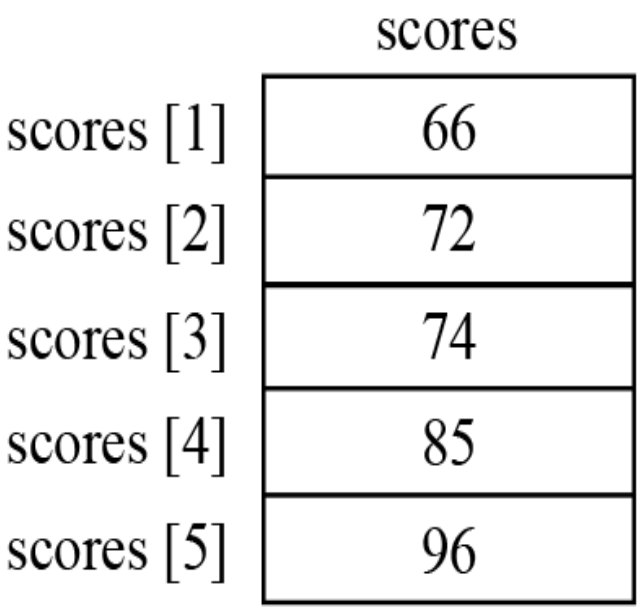
Removal of Node

```
void removeNodeAtFirst ()
{
If (pHead!=NULL)
   {
   Node *temp=pHead;
        pHead = pHead ->next;
   delete temp;
   }}
```

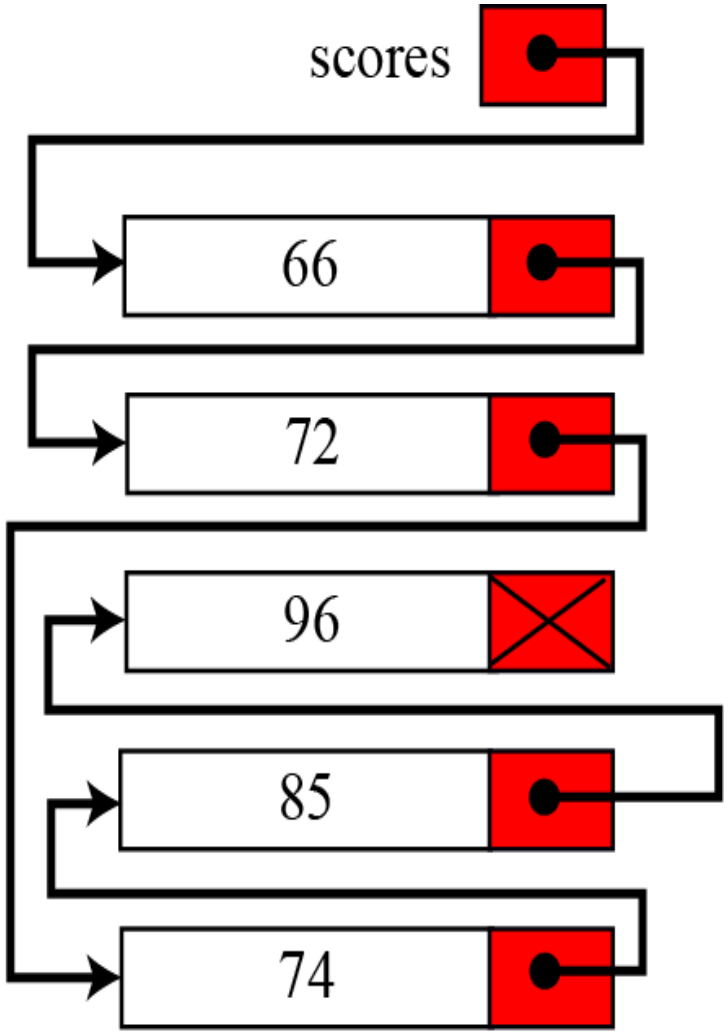
Removal of a node



Arrays versus linked lists



a. Array representation



b. Linked list representation

Contd…

- The elements of an array occupy contiguous memory locations.
- The nodes of Linked list are not constrained to be stored in adjacent locations.
- Insertions and deletions in linked list are less complex comparatively to an array.

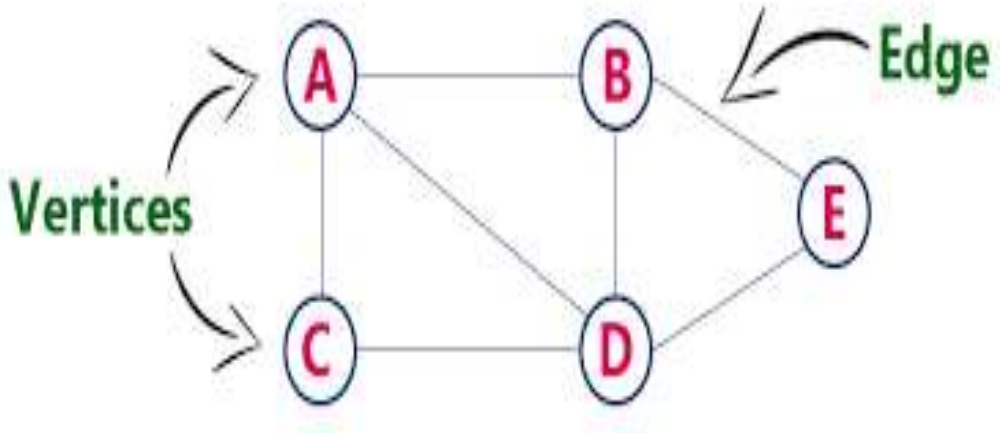
Graph

- Graph is a data structure that consists of a set of vertices and a set of edges.
- The set of edges describes the connection among vertices.
- A Graph G is defined as follows:

  G=(V,E)

  Where V is the set of vertices and E is the set of edges.

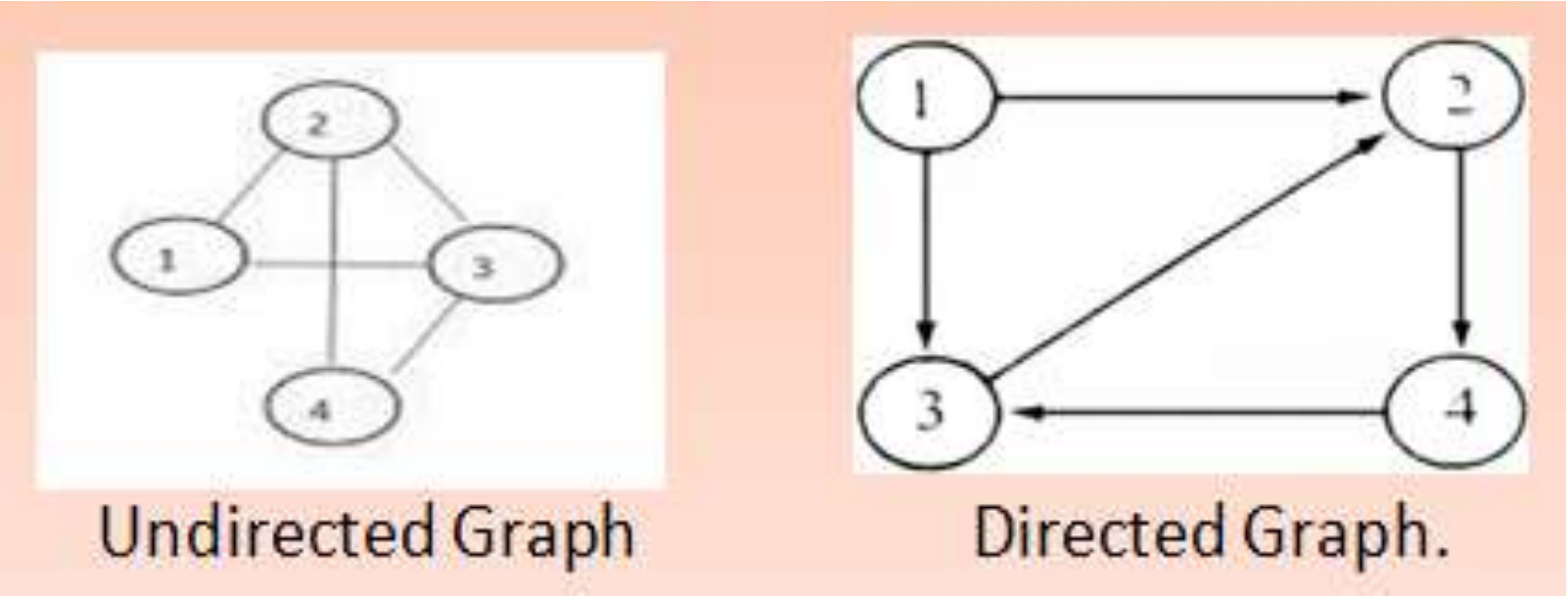
Contd…



Here set of vertices={A,B,C,D,E}
and Set of Edges={AB,AC,AD,BD,BE,CD,DE}

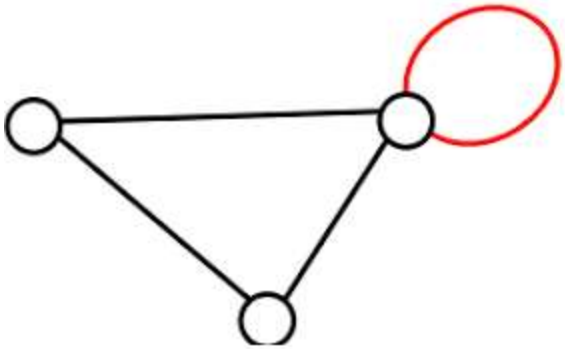
Directed Vs Undirected Graph

- Undirected Graph has no orientation or direction and all edges are undirected
- Directed Graph has an orientation and all edges are directed.



Contd…

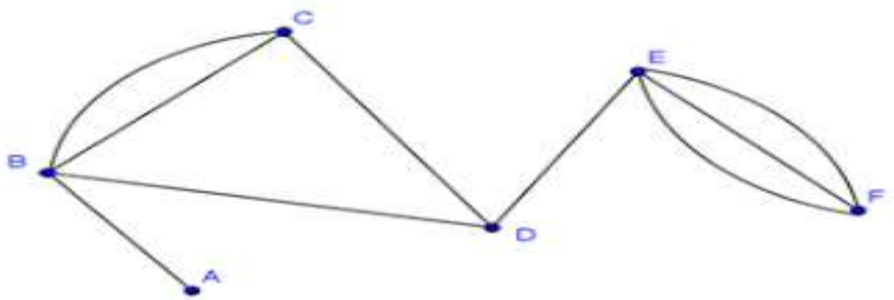
- The edge that connects a vertex to itself is called a loop.
- The sequence of vertices ,such that each adjacent pair of vertices are connected by an edge generates a path.



Graph with Loop

Contd…

- If two vertices of the graph are connected by more than one edge then the graph is known as multiple edges graph.
- Simple graph has no loop and no multiple edge.

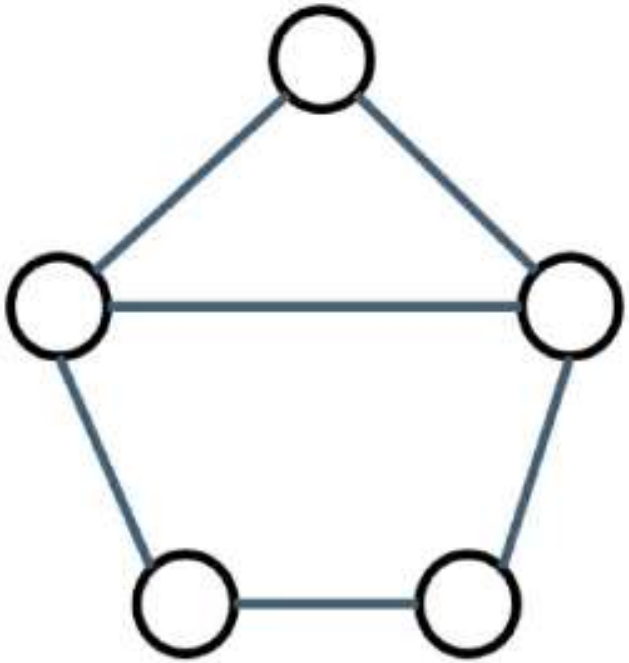


Graph with Multiple Edges

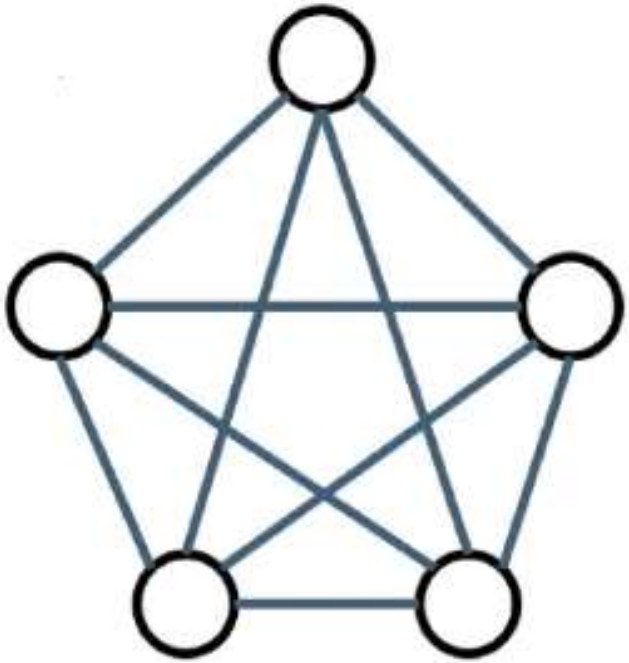
Planer Vs Non Planer Graph

- Planer Graph- A graph G is called a planar graph if it can be drawn in a plane without any edges crossed.
- Non-planar Graph – A graph is called non-planar if it cannot be drawn in a plane without graph edges crossing.

Contd…



Planar



Non-Planar

Degree of a Vertex

- Degree of a vertex is the number of edges adjacent to a vertex V and is denoted by deg(V). Here

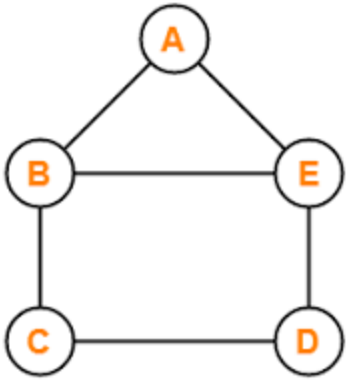
  Deg(A)=2

  Deg(B)=3

  Deg(C)=2

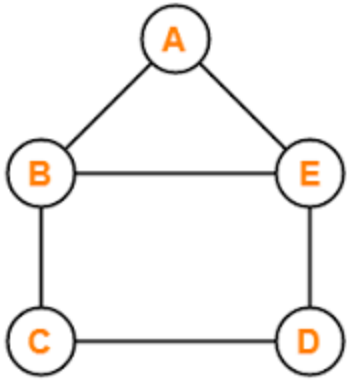
  Deg(D)=2

  Deg(E)=3



Degree of A Graph

The degree of a graph is the largest value among the degrees of all vertices. Here the degree of the the graph is 3.



Thank You