GIET
In Pursuit of Innovation

GIET POLYTECHNIC

LECTURE NOTES

ON

DATASTRUCTURE

Compiledby

Rajlaxmi Sahoo

Department of Computer Science and Engg.

CONTENTS

Chapter-1

INTRODUCTION

Data is the basic entity or fact that is used in calculation or manipulation process. There are two types of data such as numerical and alphanumerical data. Integer and floating-point numbers are of numerical data type and strings are of alphanumeric data type. Data may be single or a set of values and it is to be organized in a particular fashion. This organization or structuring of data will have profound impact on the efficiency of the program.

DATA:

Data is value or set of values which does not give any meaning. It is generally a raw fact.

For example:

1.34	3.Chintan
2.13/05/2008	4.12,34,43,21

ENTITY

An entity is a thing or object in the real world that is distinguishable from all other objects.

- The entity has a set of properties or attributes and the values of some sets of these attributes may uniquely identify an entity.
- An entity set is a collection of entities.

Example:

Entity	Student			
Attributes	Roll No.	Name	DOB	% of marks scored
Values	123	Ram	01/01/1980	79%

All students of a particular class constitute an entity set.

INFORMATION

It can be defined as meaningful data or processed data. When the raw facts are processed, we get a related piece of information as its output/result.

Example:

Data(01/01/1980)becomes information if entity Ram is related to Date of birth attribute (01/01/1980) as follows:

DOB of student Ram is 01/01/1980

DATATYPE

A data type is a term which refers to the kind of data that may appear in computation. Every programming language has its own set of built-in data types.

Example:

Data	Datatype
34	Numeric
Chintech	String
21,43,56	Array of integers
12/05/2008	Date

In C ,the following are the basic data types are: int , long, char, float, double, void etc.

Definition:

Data Structure is a specialized format for storing data so that the data's can be organized in an efficient way.

Or

"Thedatastructureisthelogicalormathematicalmodelofparticularorganizationof data.".

Or

A data structure is a mathematical or logical way of organizing data in the memory that consider not only the items stored but also the relationship to each other and also it is characterized by accessing functions.

Data Structure=Organizeddata+AllowedOperations.

OBJECTIVES OF DATA STRUCTURE

- To identify and create useful mathematical entities and operations to determine what classes of problems can be solved by using these entities and operations.
- To determine the representation of these abstract entities and to implement the abstract operations on these concrete representation.

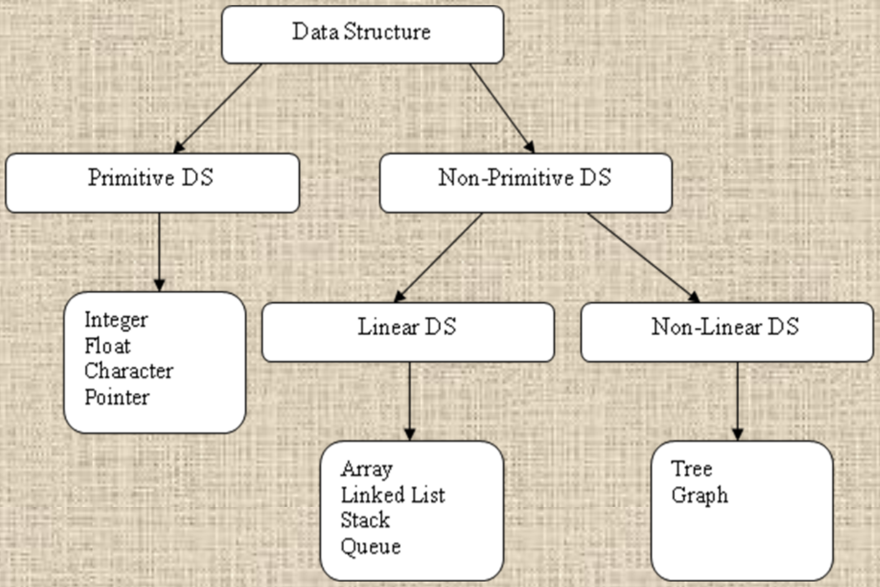
NEED OF A DATASTRUCTURE

- To understand the relationship of one data elements with the other and organize it within the memory.
- A data structures helps to analyze the data, store it and organize it in a logical or mathematical manner.
- Data structures allow us to achieve an important goal component reuse.

CLASSIFICATION OF DATA STRUCTURE

Based on how the data items are operated ,it will classify into two broad categories.

- Primitive Data Structure
- Non-Primitive Data Structure



Primitive Data Structure: These are basic DS and the data items are operated closest to the machine level instruction.
Example: integer, characters , strings, pointer and double.

Non-Primitive Data Structure: These are more sophisticated DS and are derived from primitive DS. Here data items are not operated closest to machine level instruction. It emphasize on structuring of a group of homogeneous (sametype) or heterogeneous(different type) data items.

Linear Data Structure: In which the data items are stored in sequence order.
Example: Arrays, LinkedLists, Stacks and Queues

Non Linear Data Structure: In Non-Linear data structures, the data items are not in sequence.
Example: Trees, Graphs.

DATASTRUCTURE OPERATIONS:
The various operations that can be performed on different data structures are described below:

1. **Creating**–A data structure created from data.
2. **Traversal**–Processing each element in the list
3. **Search**– Finding the location of given element.

4. **Insertion** –Adding a new element to the list.
5. **Deletion**–Removing an element from the list.
6. **Sorting**–Arranging the records either in ascending or descending order.
7. **Merging**–Combining two lists into a single list.
8. **Modifying**– the values of DS can be modified by replacing old values with new ones.
9. **Copying**– records of one file can be copied to another file.
10. **Concatenating** – Records of a file are appended at the end of another file.
11. **Splitting**– Records of big file can be splitting into smaller files.

ABSTRACT DATA TYPES

Abstract data type (ADT) is a specification of a set of data and the set of operations that can be performed on the data.

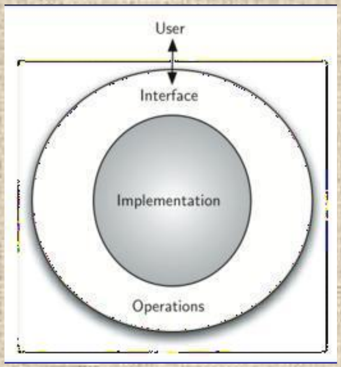
❖ It is a logical description of how we view the data and the operations that are allowed without knowing how they will be implemented.

❖ This means that we are concerned only with what the data is representing and not with how it will eventually be constructed.

❖ An ADT is a data declaration packaged together with the operations that are meaningful on the data type.

1. Declaration of Data
2. Declaration of Operations

Examples: Objects such as lists, sets and graphs with their operations can be called as ADT. For the SET ADT, we might have operations such as *union, intersection, and complement* etc.



Uses of ADT:

1. It helps to efficiently develop well designed program
2. Facilitates the decomposition of the complex task of developing a software system into a number of simpler subtasks
3. Helps to reduce the number of things the programmers has to keep in mind at anytime.
4. Breaking down a complex task into a number of earlier subtasks also simplifies testing and debugging.

ALGORITHM:

An *algorithm* is a method of representing the step by step logical procedure for solving a problem. It is a tool for finding the logic of a problem.
In addition every algorithm must satisfy the following criteria:

- **Input:** There are zero or more quantities which are externally supplied. i.e. each algorithm must take zero, one or more quantities as input data.
- **Output:** At least one quantity is produced .i.e. produce one or more output values.
- **Definiteness:** Each instruction or step of an algorithm must be clear and unambiguous.
- **Finiteness:** An algorithm must terminate in a finite number of steps.
- **Effectiveness:** Each step must be effective, in the sense that it should be primitive(easily convertible to program).

ALGORITHM COMPLEXITY:

- ➢ After designing an algorithm, we have to be checking its correctness. This is done by analyzing the algorithm. The algorithm can be analyzed by tracing all step-by-step instructions, reading the algorithm for logical correctness, and testing it on some data using mathematical techniques to prove it correct.
- ➢ There may be more than one algorithm to solve a problem. The choice of a particular algorithm depends on its complexity.
- Complexity refers to the rate at which the required storage or consumed time grows as a function of the problem size.
- It is the performance evaluation or analysis/ measurement of an algorithm is obtained by totaling the number of occurrences of each operation when running the algorithm.
- Two important ways to characterize the effectiveness of an algorithm are its space complexity and time complexity.
- ▪ Space Complexity
- ▪ Time Complexity

SPACECOMPLEXITY:

- ➢ Space complexity of an algorithm or program is the amount of memory it needs to run to completion.
- ➢ We can measure the space by finding out that how much memory will be consumed by the instructions and by the variables used.

Example:

Suppose w want to add two integer numbers .To solve this problem we have following two algorithms:

Algorithm1:	**Algorithm2:**
Step1- Input A.	Step1- Input A.
Step2- Input B.	Step2- Input B.
Step3-Set C:= A+ B.	Step3-Write: ‘Sum is‘, A+B.

Step4-Write: ‘Sum is‘, C.

Step 4- Exit. Step 5- Exit.

Both algorithms will produce the same result. But first takes 6 bytes and second takes 4 bytes (2bytesforeachinteger).And first has more instructions than the second one. So we will choose the second one as it takes less space than the first one. (Space Complexity).

Suppose 1 second is required to execute one instruction. So the first algorithm will take 4 seconds and the second algorithm will take 3 seconds for execution. So we will choose the second one as it will take less time. (Time Complexity).

TimeComplexity

- The time complexity of a program /algorithm is the amount of computer time that is need to run to completion.
- But the calculation of exact amount of time required by the algorithm is very difficult. So we can estimate the time and to estimate the time we use some asymptotic notation, which are described below. Let the no. of steps of an algorithm is measured by the function f(n).

1. Big ‘O’ notation

   The given function f(n) can be expressed by big ‘O’ notation as follows .f(n)= O(g(n)) if and only if there exist the +ve const. ‘C’ and no such that f(n) ≤ C * g(n) for all n ≥ no.

2. Omega(Ω)notation
   The function f(n)=Ω(g(n)) if and only if there exist the +ve const C and no such that f(n)≥C * g(n) for all n≥ no

3. Theta(θ)notation:
   The function f(n)=θ(g(n)) if and only if there exist a +ve const‘C1’,‘C2 ’ and no such that C1 * g(n)≤f(n) ≤C2*g(n)

   Big ‘O’ is the upper bound Ω is the lower bound θ is the avg bound which can be estimated in time complexity.

SpaceComplexity

The space complexity is the program that the amount of memory that is needed to run to completion .The space complexity need by a program have two different Components.

1. Fixed space requirement: -The components refer to space requirement that do not depend upon the number and size of the programs input and output.

2. Variable space requirement: - This component consist of the space needed by structure variable whose size depends on the particular instruction, of the program being solved.

TIME-SPACETRADE-OFF:

- There may be more than one approach (or algorithm) to solve a problem.
- The best algorithm (or program) to solve a given problem is one that requires less space in memory and takes less time to complete its execution.
- But in practice ,It is not always possible to achieve both of these objectives.
- One algorithm may require more space but less time to complete its execution while the other algorithm requires less time space but takes more time to complete its execution.
- Thus,we may have to sacrifice one at the cost of the other.
- If the space is our constraint, then we have to choose a program that requires less space at the cost of more execution time.
- On the other hand, if time is ou constraint such as in real time system, we have to choose a program that takes less time to complete its execution at the cost of more space.

Chapter3ARRAYS

An array is a finite, ordered and collection of homogeneous data elements.

finite – It contain limited no. of element.

ordered–All the elements are stored one by one in a contiguous memory location.

homogeneous -All the elements of an array of same type.

The elements of an array are accessed by means of an index or subscript.

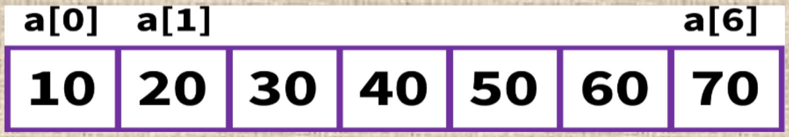
That’s why array is called subscripted variable.

LINEARARRAY

If one subscript is required to refer all the elements in an array then this is called Linear array or one-dimensional array**.**

Representation of Linear Array in memory

Let a is the name of an integer array.
It contains a sequence of memory location**.**



Let b=address of the 1st element in the arrayi.e. base address If w=wordsize then

address of any element in array can be obtained as address of

(a[i]) = b + i x w   i=index   no.addressof

a[3] = b + i x w

=b+3x2=b+6

Operation on Array

1. TRAVERSAL

This operation is used to visit all the elements of the array.

```
Voidtraverse(inta[],intn)
{
inti;
for(i=0;i<n;i++)
Printf("%d",a[i]);
}
```

2. INSERTION

Inserting the array at the end position can be done easily, but to insert at the middle of the array we have to move the element to create a vacant position to insert the new element.

```
int insert (int a[], int n, int pos, int ele)
{
inti;
for(i=n-1;i>=pos-1;i--)
{
a[i+1]=a[i];
}
a[pos-1]=ele; n++;
return(n);
}
```

3. DELETION

Deleting an element at the end of the array can be done easily by only decreasing the array size by 1.

But deleting an element at the middle of the array require that each subsequent element from the position where to be deleted should be moved to fill up the array.

```
intdelete(int a[],int n,int pos)

{
```

```
for (i= pos-1;i<n-1;i++)

{

a[i]=a[i+1];

} n--;

return(0);

}
```

Memory Representation of 2DArray

- The array having two subscript is called as 2Darray.
- In 2Darray the elements are stored in contiguous memory location as in single dimensional array.
- There are 2 ways of storing any matrix in memory.
  1. Row-major order
  2. Column-major order

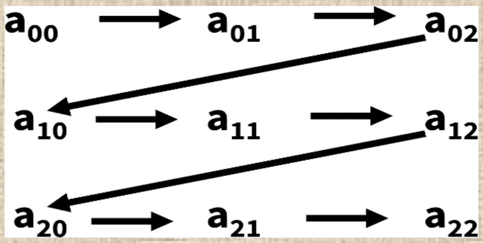
Row-major Order

In row-major order the row elements are focused first that means elements of matrix are stored on a row-by-row basis.

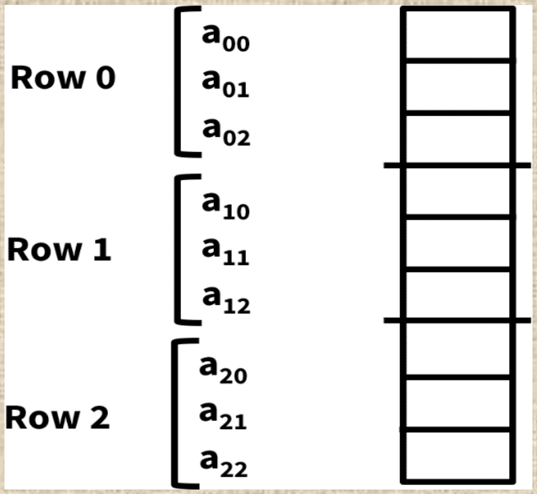
LogicalRepresentationofarraya[3][3]

$$\begin{bmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{bmatrix}$$

Row major order representation of a[3][3]



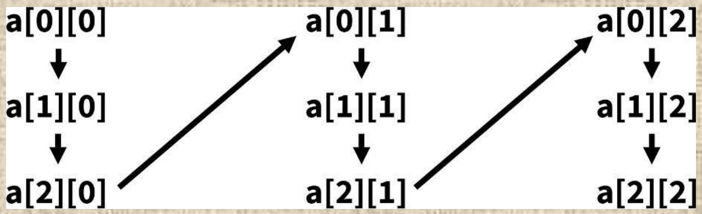
Storage Representation in Row-major order



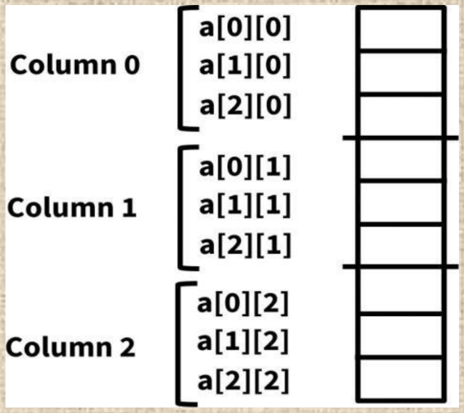
Column-major Order

In column major order the column elements are focused first that means elements of the matrix are stored in column-by-column basis.

Ex:Column major order representation of a[3][3]



Storage Representation of matrix inColumn-majorOrder



Address Calculation of Matrix in Memory

Row-major order

Suppose a[u1][u2]is a 2Darray

u1=no.of row

u2 = no. of column
Address of a[i][j]=b+(i * u2+ j)*w

Column-major order

Address of a[i][j]= b+ (j * u1+i)* w

Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer.

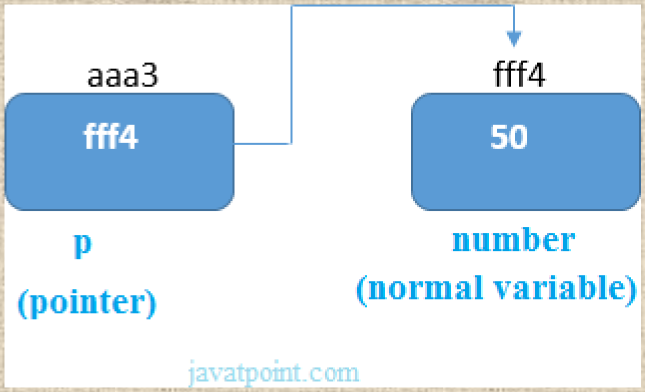
Syntax:datatype*var_name;

Example:

1. **int***a;//pointer to int
2. **char***c;//pointer to char

Pointer Example

An example of using pointers to print the address and value is given below.



In the above figure, pointer variable stores the address of number variable, i.e.,fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of*(**indirection operator**), we can print the value of pointer variable p.

Let's see the pointer example as explained for the above figure.

```
        #include<stdio.h>
        int main(){
        int number=50;
        int *p;
        p=&number;//stores the address of number variable
        printf("Address of p variable is %x \n",  p);
        printf("Value of p variable is %d \n",*p);
        return0;
}
```

Output

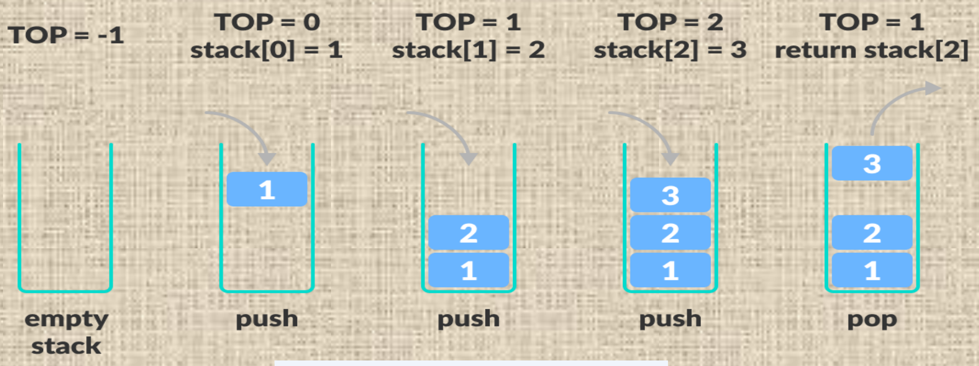
```
Address of number variable isfff4
Address of p variable is fff4
Value of p variable is 50
```

Chapter-4

STACKS&QUEUES

- Stack is a linear data structure in which an element may be inserted or deleted at one end called TOP of the stack.
- That means the elements are removed from a stack in the reverse order of that in which they were inserted into the stack.
- Stack is called LIFO(Last-in-first-Out) Str.i.e. the item just inserted is deleted first.
- There are 2 basic operations associated with stack

  1. Push:-This operation isused to insert an element into stack.
  2. Pop:-This operation isused to delete an element from stack



Working of Stack Data structure

Condition also arise:

1. Overflow:-When a stack is full and we are attempting a push operation, overflow condition arises.
2. Underflow:-When a stack is empty ,and we are attempting a pop operation then underflow condition arises.

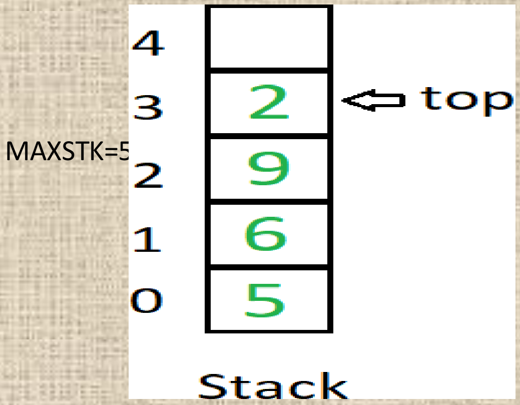
Representation of Stack in memory

A stack may be represented in the memory in two ways:

1. Using one dimensional array i.e.Array representation of Stack.
2. Using single linked list i.e.Linked list representation of stack.

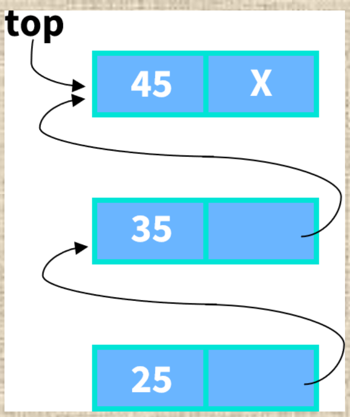
Array Representation of Stack :

To implement a stack in memory, we need a pointer variable called TOP that hold the index of the top element of the stack, a linear array to hold the elements of the stack and a variable MAXSTK which contain the size of the stack.



Linked List Representation of Stack:

- Array representation of Stack is very easy and convenient but it allows only to represent fixed sized stack.
- But in several application size of the stack may very during program execution, at that cases we represent a stack using linked list.
- Single linkedlist is sufficient to represent any Stack.



- Here the 'info' field for the item and 'next' field is used to print the next item.

INSERTION IN STACK (PUSH OPERATION)

This operation is used to insert an element in stack at the TOP of the stack.

Algorithm :-

```
PUSH(STACK,TOP,MAXSTK, ITEM)
This procedure pushes an item into stack.
1. If TOP =0 then print Underflow and Return.
 2. Set ITEM=STACK[TOP] (Assigns TOP element to ITEM)
3. Set TOP=TOP-1 (Decreases TOP by1)
4. Return
```

(Where Stack=Stack is a list of linear structure
TOP=pointer variable which contain the location of Top element of the stack
MAXSTK= variable which contain size of the stack

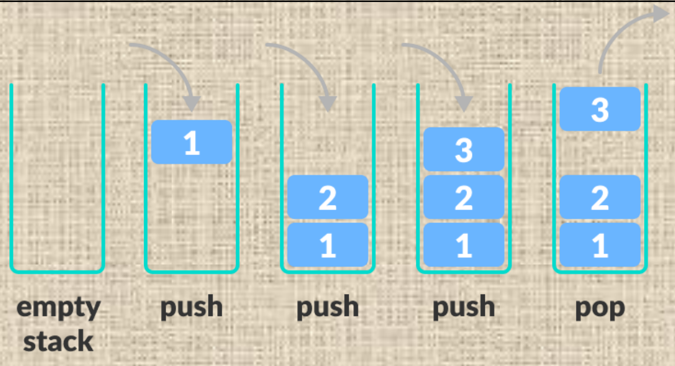
ITEM= Item to be inserted)

DELETION IN STACK (POP OPERATION)

This operation is used to delete an element from stack.

Algorithm :-

```
POP(STACK,TOP,ITEM)
This procedure deletes the top element of STACK and assigns it to the variable  ITEM.
        1. If TOP =0 then print Underflow and Return.
        2. Set ITEM=STACK[TOP] (Assigns TOP element to ITEM)
        3. Set TOP=TOP-1(Decreases TOP by1)
        4. Return
```

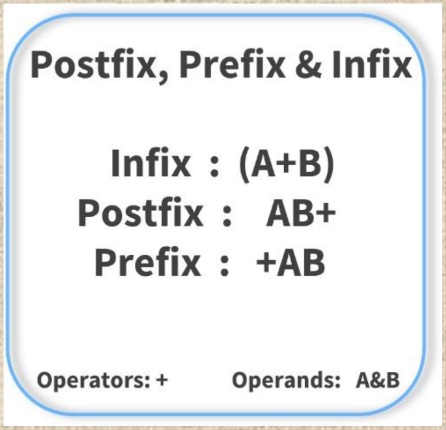


Stack Push and Pop Operations

<u>ARITHMETICEXPRESSION</u>

There are 3 notation to represent an arithmetic expression.

1. Infix notation
2. Prefix notation
3. Postfix notation



Postfix, Prefix & Infix

Infix : (A+B)
Postfix : AB+
Prefix : +AB

Operators: + Operands: A&B

1. **<u>INFIX NOTATION</u>**
   - The convention always of writing an expression is called as infix. Ex:A+B, C+D, E*F, G/M etc.
   - Here the notation is <Operand><Operator><Operand>
   - This is called infix because the operators come in between the operands.
2. **<u>PREFIX NOTATION</u>**
   - This notation is also known as "POLISH NOTATION"
   - Here the notation is <Operator><Operand><Operand>
   - Ex:+AB,-CD,*EF, /GH

3. **POSTFIX NOTATION**
    - This notation is called as post fix or suffix notation where operator is suffixed by operand.
    - Here the notation is <Operand><Operand><Operator>
    - Ex:AB+, CD-,EF*,GH/
    - This notation is also known as “REVERSEPOLISH NOTATION.”

CONVERSIONFROM INFIXTOPOSTFIXEXPRESSION

Algorithm:

POLISH(Q,P)

Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

```
1. Push“(” onto stack and add“)”to the end of Q.
2. Scan Q  from left to right and repeat steps3to6 for each element of Q until the STACK is
   empty.
3. If an operand is encountered, add it top.
4. If a left parenthesis is encountered, push it onto stack.
5. If an operator X is encountered,then:
       a) Repeatedly POP from STACK and add to P each operator (on the top of the
          STACK) which has the same precedence as or higher precedence than the
          operator X .
       b) Add the operator X to STACK.
6. If a right parenthesis is encountered then:
       a) Repeatedly POP from STACK and add to P each operator( on the top of the
          STACK) until a left parenthesis is encountered.
       b) Remove the leftparenthesis.
[End of if str.]
       [EndofStep-2Loop]
7. Exit
```

Ex:A+(B*C–(D/E^F)*G)*H)

SymbolScanned	STACK	EXPRESSION(POSTFIX)
	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
C	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
^	(+(-(/^	ABC*DE
F	(+(-(/^	ABC*DEF
)	(+(-	ABC*DEF^/
*	(+(-*	ABC*DEF^/
G	(+(-*	ABC*DEF^/G
)	(+	ABC*DEF^/G*-
*	(+*	ABC*DEF^/G*-
H	(+*	ABC*DEF^/G*-H
)		ABC*DEF^/G*-H*+

Equivalent Postfix Expression: ABC*DEF^/G*-H*+

EVALUATION OF POSTFIX EXPRESSION

Algorithm:

This algorithm finds the value of an arithmetic expression P written in Postfix notation.

1. Add a right parenthesis“) ”at the end of P.
2. Scan P from left to right and repeat steps 3 and 4 for each element of P until the “) ” is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator X is encountered then:
   a) Remove the two top element of STACK, where A is top element and B is the next-to-top element.

```
        b) Evaluate BXA.
        c) Place the result of(b) on STACK.
[End of if str.]
    5. Set value equal to the top element on STACK.
    6. Exit
```

Ex: 5, 6,2,+, *, 12,4, /,-)

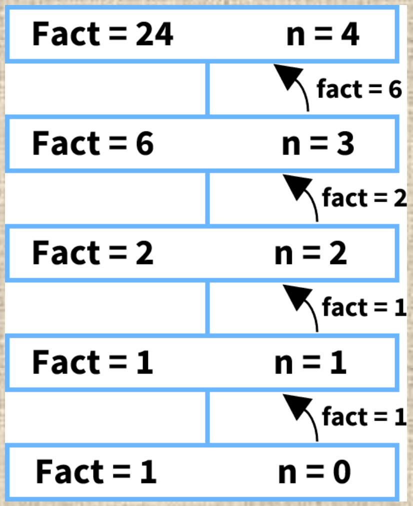
SymbolScanned	STACK
5	5
6	5,6
2	5,6,2
+	5,8
*****	40
12	40,12
4	40,12,4
/	40,3
-	37(Result)
)	

APPLICATIONSOFSTACK:

- Recursion process uses stack for its implementation.
- Quick sort uses stack for sorting the elements.
- Evaluation of arithmetic expression can be done by using STACK.
    - Conversion from infix to postfix expression
    - Evaluation of postfix expression
    - Conversion from infix to prefix expression
    - Evaluation of prefix expression.
- Backtracking
- Keep track of post-visited(history of a web-browsing)

IMPLEMENTATION OF RECURSION

A function is said to be Recursive function if it call itself or it call a function such that the function call back the original function.
This concept is called as Recursion.
The Recursive function has two properties.



I. The function should have a base condition.
II. The function should come closer towards the base condition.

The following is one of the example of recursive function which is described below.

Factorial of a no.using Recursion

The factorial of a no. ‘n’ is the product of positive integers from1to n.

n ! = 1 X 2 X 3 X ....X (n-) X n
Physically provedn!= nX (n-1)!

The factorial function can be defined as follows.

I. If n=0then n!=1
II. If n>0then n!=n.(n-1)!

The factorial algorithm is given below factorial(fact,n)

This procedure calculates n! and returns the value in the variable fact.

```
1. If n=0 then fact=1and Return.
2. Call factorial(fact,n-1)
3. Set fact= fact*n
4. Return
```

Ex: Calculate the factorial of 4.

4 ! = 4 x 3 !

3!= 3x2!

2!=2x1!

1!=1x0!

0!=1

1!=1x1=1

2!=2x1=2

3!=3x2=6

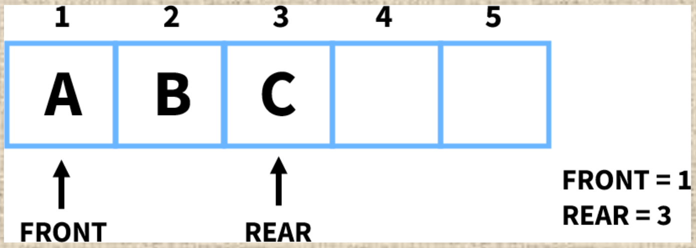
4!=4x6=24

QUEUE

- Queue is a linear data structure or sequential data structure where insertion take place at one end and deletion take place at the other end.
- The insertion end of Queue is called rear end and deletion end is called front end.
- Queue is based on (FIFO) First in First Out Concept that means the node i.e. added first is processed first.
- Here Enqueue is Insert Operation.
- Dequeue is nothing but Delete Operation.

Representation Of Queue In Memory

- The Queue is represented by a Lineararray “Q” and two pointer variable FRONT and REAR.
- FRONT gives the location of element to be deleted and REAR gives the location after which the element will be inserted.
- The deletion will be done by
  setting Front = Front + 1
- The insertion will be done by
  setting Rear = Rear + 1



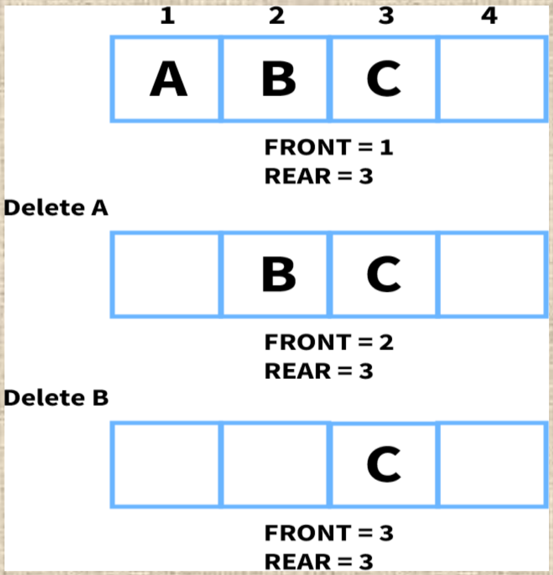
INSERTIONINQUEUE(Enqueue)

Algorithm:

Insert(Q, ITEM,Front,Rear)
This procedure insert ITEM in queue Q.

```
1. If Front=1andRear=Max
   Print ‘Overflow’ and Exit.
2. If  front=NULL  then
       Front = 1
       Rear=1
   else
```

```
        Rear =Rear+1
3.  Q[Rear]= ITEM
4.  Exit
```

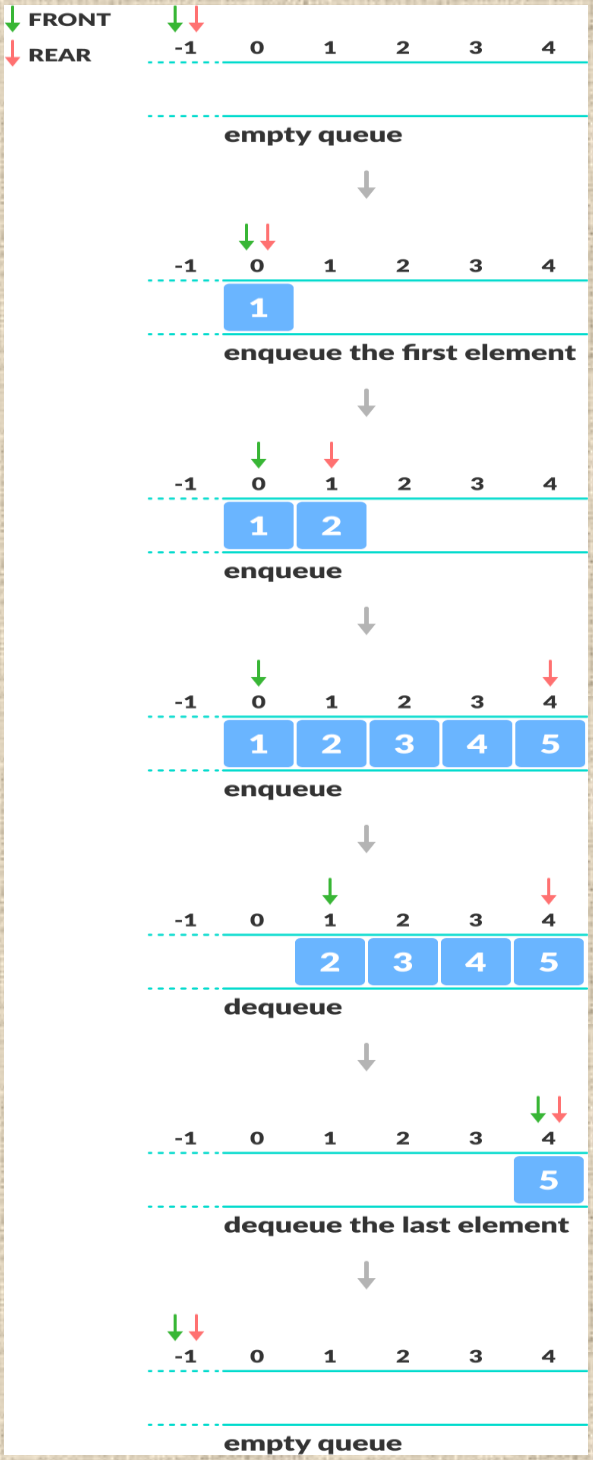


DELETIONIN QUEUE(Dequeue)

Algorithm:Delete(Q,ITEM, FRONT, REAR)
This procedure remove element from queue Q.

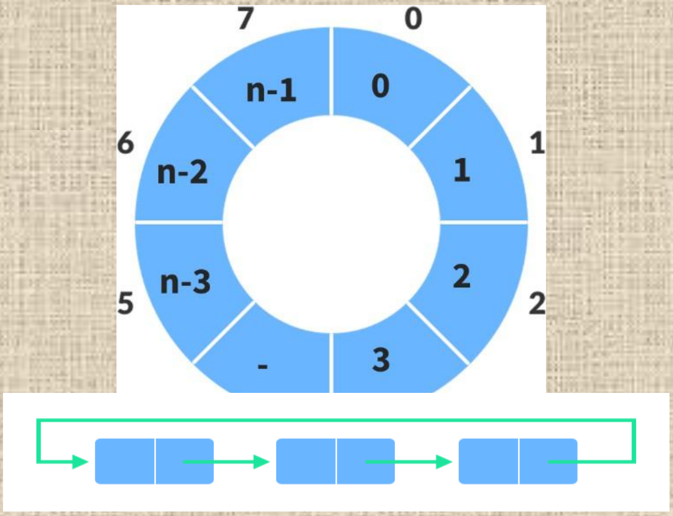
```
1.  If    Front=Rear=NULL
Print 'Underflow' and Exit.
2.  ITEM=Q(FRONT)
3.  If   Front=Rear
        Front =NULL
        Rear = NULL
    else
        Front =Front+1
4.  Exit
```

EnqueueandDequeue Operations



CIRCULARQUEUE

- Let we have a queue that contain ‘n’ elements in which Q[1] comes after Q[n].
- When this technique is used to construct a queue then the queue is called circular queue.
- In Circular queue when REAR is ‘n’ and any element is inserted then REAR will set as 1.
- Similarly when the front is n and any element is deleted then front will be 1.



Circular Queue representation

INSERTION ALGORITHM OF CIRCULARQUEUE

```
Insert(Q,N,FRONT, REAR, ITEM)
This procedure inserts an element ITEM into the circular queue Q.
    1. If Front=1and Rear=N
then print ‘Overflow’ and Exit.
    2. If front=NULL
  Then Front=1andRear=1 else
if Rear = N then
        Set Rear =1
else
        set Rear=Rear+1
        (End of if Str.)
    3. Set Q[Rear]=ITEM(Insert a new element)
    4. Exit
```

DELETION ALGORITHM OF CIRCULARQUEUE

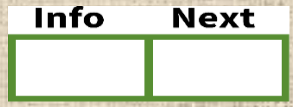
```
Delete(Q, N,ITEM,FRONT,REAR)
This procedure delete an element from a circular queue.
      1.  If Front=NULL then write Underflow and Return.
      2.  Set ITEM=Q[FRONT]
      3.  If Front= Rear
Then Front=NULL and Rear=NULL Else
if Front = N then
            Set Front=1
else
         Front= Front+1
   [End of if str.]
      4.  Return
.
```

Chapter-5

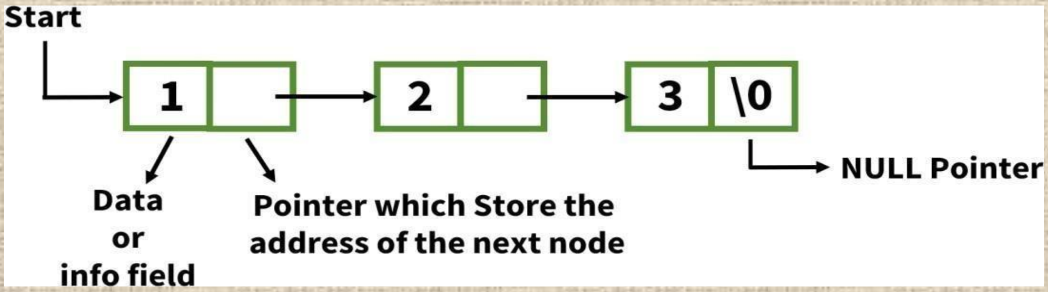
LINKEDLIST

Linked List is a collection of data elements called as nodes.
The node has 2 parts

- Info is the data part
- Next i.e. the address part that means it points to the next node.



If linked list adjacency between the elements are maintained by means of links or pointers.
A link or pointer actually the address of the next node.



The last node of the list contain'\0'NULLwhichshows the end of the list.
The linked list contain another pointer variable 'Start' which contain the address the first node.
Linked List is of 4 types

- Single Linked List
- Double Linked List
- Circular Linked List
- Header Linked List

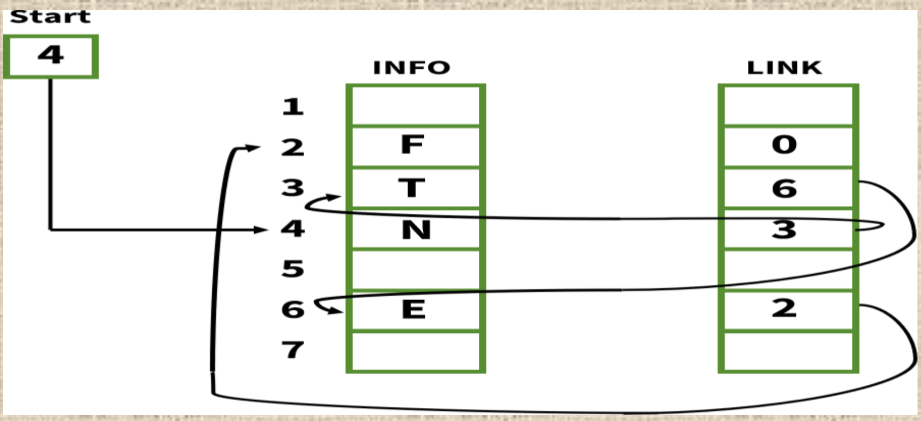
Representation of Linked List in Memory

- Linked List can be represented in memory by means 2 lineararrays i.e.Data or info and Link or address.

- They are designed such that info[K] contains the Kth element and Link[K] contains the next pointer field i.e. the address of the Kth element in the list.

- The end of the List is indicated by NULL pointer i.e. the pointer field of the last element will be NULL or Zero

.



Start=4	
info[4]=N	Link[4]=3
info[3]=T	Link[3]=6
info[6]=E	Link[6]=2
info [2] = F	Link[2]=0i.e.the null value

So the list has ended .

Representation of a node in a LinkedList

```
Struct Link
{
int data;
Struct Link *add;
};
```

SINGLE LINKEDLIST

A Single Linked List is also called as one-way list. It is a linear collection of data elements called nodes, where the linear order is given by means of pointers.

Each node is divided into 2 parts.

I. Information

II. Pointer to nextnode.

OPERATION ON SINGLELINKEDLIST

1. Traversal

Algorithm:

```
Display(Start)This  algorithm traverse the list starting from the1st node to the end of the list.
Step-1 : "Start" holds the address of the first node.
Step-2 : Set Ptr = Start [Initializes pointer Ptr]
Step-3: Repeat Step 4to5, WhilePtr ≠NULL
Step-4: Process info[Ptr] [apply Process to info(Ptr)]
Step-5:Set  Ptr=next[Ptr]  [movePtr  to  next  node]
[End of loop]
Step-6:Exit
```

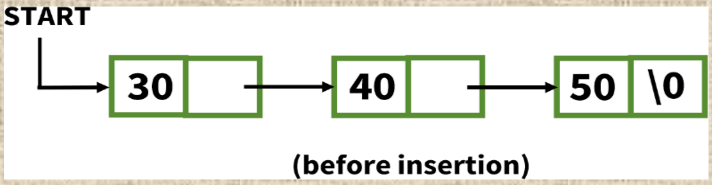
2. Insertion

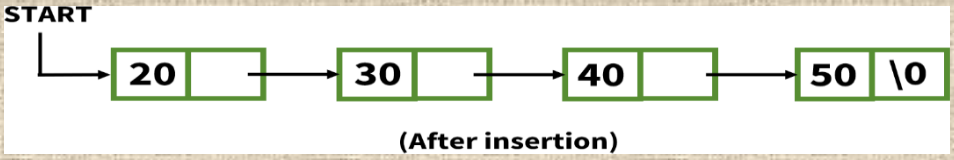
The insertion operation is used to add an element in an existing Linked List. There is various positions where node can be inserted.

- Insert at the beginning
- Insert at end
- Insert at specific location.

Insert At The Beginning

Suppose the new node whose information field contains 20 is inserted as the first node.



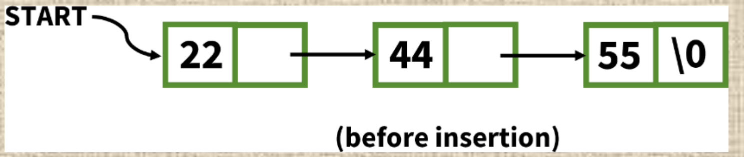


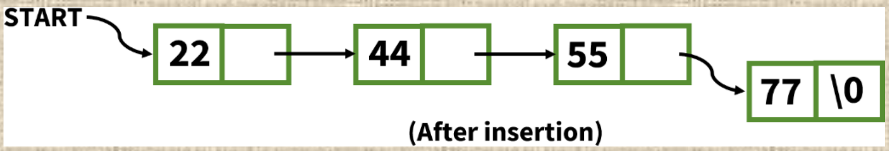
Algorithm:
This algorithm is used to insert a node at the beginning of the Linked List. Start holds the address of the first node.

```
Step-1:Create a new node named as P.
Step-2:If   P   ==NULL   then   print“Outofmemoryspace”andexit.
Step-3 : Set info [P] = x (copies a new data into a new node)
Step-4:Set next[P]=Start(new   node now points to original first node)
Step-5 : Set Start = P
Step-6:Exit
```

Insert At The End

To insert a node at the end of the list, we need to traverse the List and advance the pointer until the last node is reached.
Suppose the new node whose information field contains 77 is inserted at the last node.





(After insertion)

Algorithm:

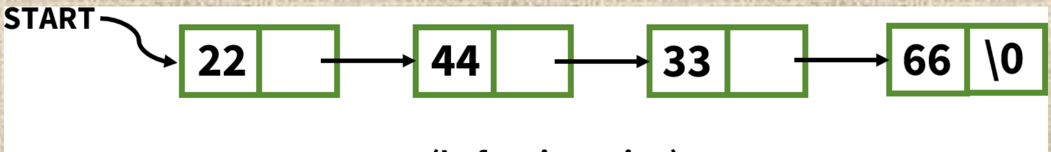
This algorithm is used to insert a node at the end of the linked list. 'Start' holds the address of the first node.

```
Step-1 : Create a new node named as P.
Step-2:If P =NULL then print"Out of memory space"andExit.
 Step-3 : Set info [P] = $x$ (copies a new data into a new node)
Step-4:Set next[P]=NULL
Step-5 : Set Ptr = Start
Step-6: Repeat Step-7whilePtr≠NULL
Step-7 : Set temp = Ptr
            Ptr=next[Ptr](Endofstep-6loop)
Step-8 : Set next [temp] = P
 Step-9:Exit
```

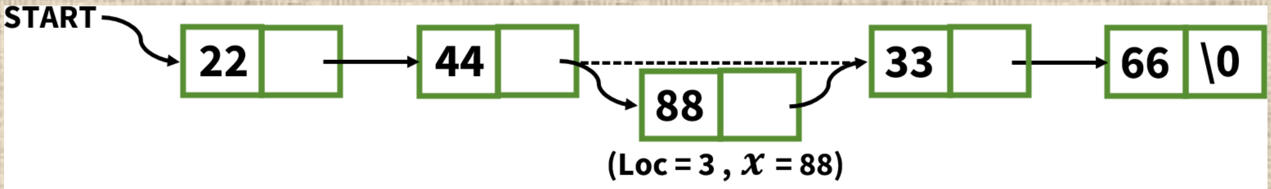
<u>Insert At Any Specific Location</u>

To insert a new node at any specific location we scan the List from the beginning and move up to the desired node where we want to insert a new node.

In the below fig. Whose information field contain 88 is inserted at 3rd location.



(before insertion)



(after insertion)

Algorithm:

'Start' holds the address ofthe1st node.

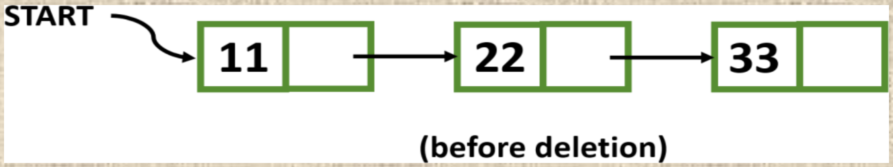
```
Step-1 : Set Ptr = Start
Step-2:Create a new node named as P.
Step-3 : If P = NULL then write'Out of memory' and Exit.
Step-4:Set info[P]= $x$ (copies a newdata into a newnode)
Step-5 : Set next [P] = NULL
```

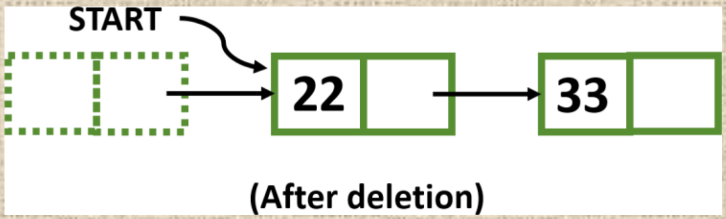
```
Step-6:Read Loc
Step-7:Seti= 1
Step-8:Repeat steps 9to11 while Ptr ≠NULL and i<Loc
Step-9 : Set temp = Ptr.
Step-10:Set Ptr=next [Ptr]
Step-11:Set i=i+1
[End of step-7 loop]
Step-12:Set next[temp]=P
Step-13 : Set next [P] = Ptr
Step-14 : Exit
```

3. Deletion

The deletion operations used to delete an element from a single linked list. There are various position where node can be deleted.

Deletethe1st Node



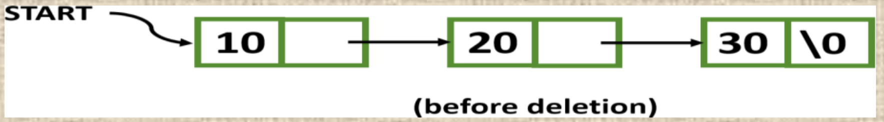


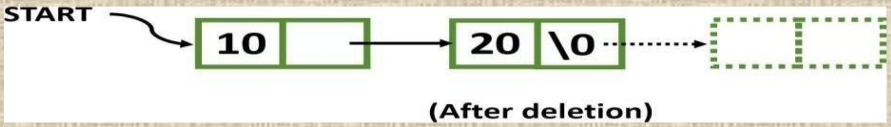
Algorithm:

Start holds the address of the 1stnode.

```
Step-1 : Set temp = Start
Step-2:If Start=NULL then  write‘UNDERFLOW’&Exit.
Step-3 : Set Start = next[Start]
Step-4:Free  the space associated with temp.
 Step-5 : Exit
```

Delete the last node

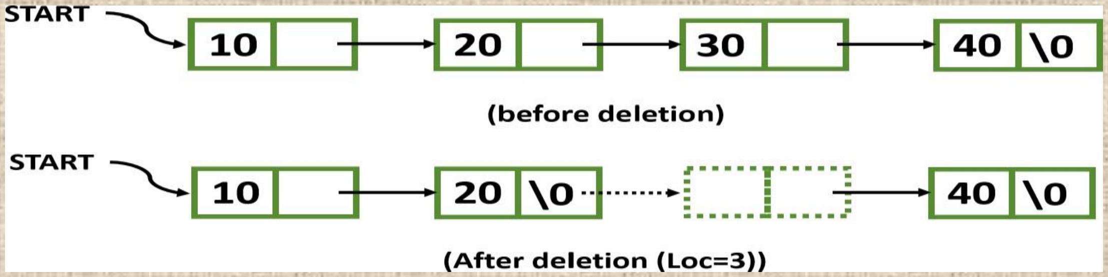




Algorithm:
Start holds the address of the 1st node.

```
Step-1:Set Ptr=Start
Step-2: Set temp=Start
Step-3: If Ptr=NULL then   write'UNDERFLOW'&Exit.
Step-4:Repeat Step-5 and 6 While next[Ptr] ≠ NULL
 Step-5 : Set temp = Ptr
Step-6:SetPtr=next[Ptr](Endofstep4loop)  Step-
7 : Set next[temp] = NULL
Step-8:Free the space associated with Ptr.
Step-9 : Exit
```

<u>Delete the node at any specific location</u>

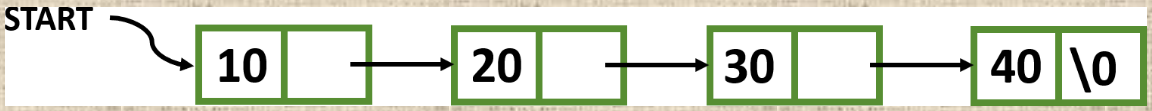


Algorithm:
Start holds the address of the1stnode.

```
Step-1 : Set Ptr = Start
Step-2:Set temp=Start
Step-3:If Ptr=NULL thenwrite'UNDERFLOW'andExit.
Step-4 : Set i = 1
Step-5:Read Loc
Step-6:Repeat Step-7to9  whilePtr ≠NULLandi<Loc
 Step-7 : Set temp = Ptr
Step-8: Set Ptr= next[Ptr]
Step-9  :  Set  i  =  i+1
(EndofStep6loop)
Step-10: Set  next[temp]= next[Ptr]
Step-11 :Free  the space associated withPtr.
Step-12 : Exit
```

<u>4. SEARCHING</u>

Searching means finding an element from a given list.



Algorithm:

```
Start  holds the address of the1$^{st}$node.
Step-1 : Set Ptr = Start
Step-2  :  Set  Loc  =  1
 Step-3:Read element
Step-4:Repeat Step-5and7 WhilePtr≠NULL
Step-5:If  element=info[Ptr]   then  Write‘Elementfound  at  position’,  Loc  and  Exit.
Step-6 : Set Loc = Loc+1
Step-7:Set Ptr=next[Ptr]
(End of step 4 loop)
Step-8:Write‘  Element  not  found’
 Step-9 : Exit
```

Headerlinkedlist

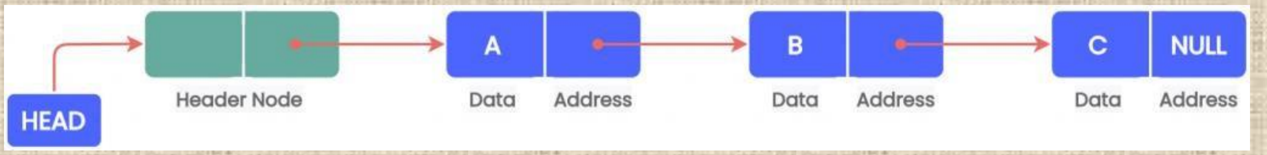
Header Linked List is a modified version of Singly LinkedList.

In Header linked list, we have a special node, the Header Node present at the beginning of the linked list.

The Header Node is an extra node at the front of the list storing meaningful information about the list.

It does not represent any items of the list like other nodes rather than the information present in the Header node is global for all nodes such as Count of Nodes in a List, Maximum among all Items, Minimum value among all Items etc.

This gives useful information about the Linkedlist.



This data part of this header node is generally used to hold any global information about the entire linked list. The next part of the header node points to the first node in the list.

A header linkedlist can be divided into two types:

i) Grounded header linked list that stores NULL in the last node’s next field.

ii) Circular header linked list that stores the address of the header node in the next part of the last node of the list.

GarbageCollection

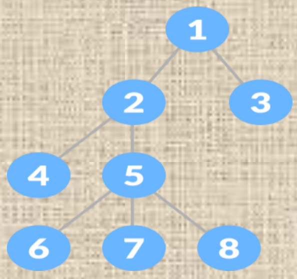
The operating system of a computer may periodically collect all the deleted space onto the free storage list. Any technique which does these collections is called garbage collection.

When we delete a particular note from an existing linked list or delete the linked list the space occupied by it must be given back to the free pool.So that the memory can be the used by some other program that needs memory space.

Chapter-6

TREE

A tree is non-linear and a hierarchical data structure consisting of a collection of nodes such that each node of the tree stores a value.



Application:

The following are the applications of trees:

- **Storing naturally hierarchical data:** Trees are used to store the data in the hierarchical structure. For example, the file system.
- **Organize data:** It is used to organize data for efficient insertion,deletionand searching.
- **Trie:**It is a special kind of tree that is used to store the dictionary.It is a fast and efficient way for dynamic spell checking.
- **Heap:**It is also a tree data structure implemented using arrays.It is used toimplement priority queues.
- **B-TreeandB+Tree:**B-Tree and B+Tree are the tree data structures used to implement indexing in databases.
- **Routingtable:** The tree data structure is also used to store the data in routing tables in the routers.

BasicTreeTerminologies

1.Node

A node is an entity that contains a key or value .

The last nodes of each path are called leaf nodes or external nodes that do not contain a link/pointer to child nodes.

The node having at least a child node is called an internal node.

2. Root Node: The topmost node of a tree or the node which does not have any parent node is called the root node. {**1**} is the root node of the tree. A non-empty tree must contain exactly one root node .

3. Parent Node: The node which is a predecessor of a node is called the parent node of that node. {**2**} is the parent node of {**6, 7**}.

4.Child Node: The node which is the immediate successor of a node is called the child node of that node. Examples: {**6, 7**} are the child nodes of {**2**}.

5. **Degree**of a Node: The total count of sub trees attached to that node is called the degree of the node. The degree of a leaf node must be 0. The degree of a tree is the degree of its root. The degree of thenode{3} is3. Lea fNode or External Node:The nodes which do not have any child nodes are called leaf nodes. {6, 14, 8, 9, 15, 16, 4,11, 12, 17, 18, 19} are the leaf nodes of the tree

6. **Sibling**:Children of the same parent node are called siblings.{8,9,10}are called siblings.

7. **Depth of anode**:The count of edges from the root to the node. Depth of node{14}is 3.

8. **Height of a node**:The number of edges on the longest path from that node to a leaf. Height of node {**3}** is **2**.

9. **Height of a tree:** The height of a tree is the height of the root node i.e the count of edges from the root to the deepest node. The height of the above tree is **3**.

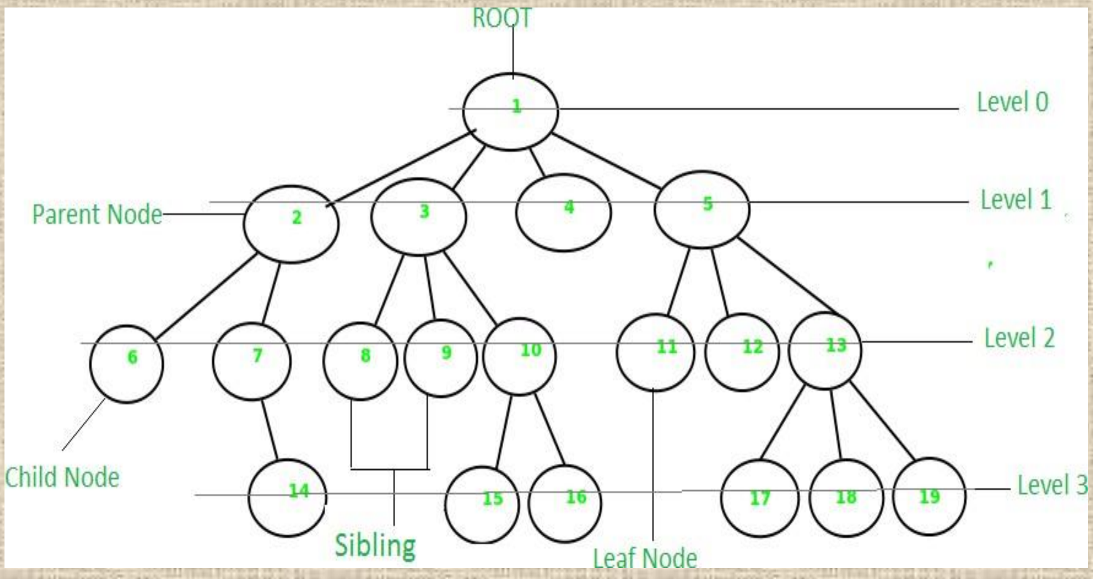
10. **Level of a node:** The count of edges on the path from the root node to that node.The root node has level **0**.

11. **Internal node:** A node with atleast one child is called Internal Node.

12. **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node. The node 1 is called ancestor of node7 as there is successive parent present from node 7 to node 1. {**1, 2}** are the parent nodes of the node {**7}**

13. **Descendant:** Any successor node on the path from the leaf node to that node.{**7,14}**are the descendants of the node. {**2}**.

14. **Path**:A sequence of edges is called as path.



BinaryTree

Each node of a binary tree consists of three items:

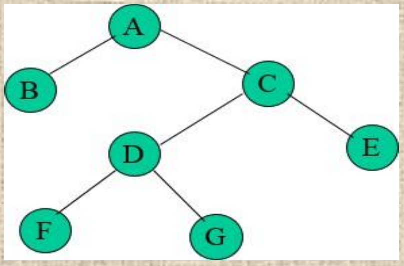
- Data item
- Address of left child
- Address of right child

A binary tree is a tree-type non-linear data structure with a maximum of two children for each parent. Here, binary name itself suggests that 'two';therefore, each node can have either 0, 1or2 children.

TypesofBinary Tree

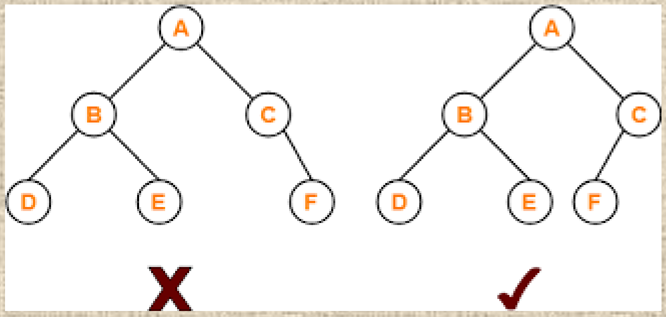
1. **Full/proper/strict Binarytree**

The full binary ree is also known as a strict binary tree.The tree can only be considered as the full binary tree if each node must contain either 0or2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.



2. **Complete Binary tree**

The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level,all the nodes must be as left as possible. In a complete binarytree, the nodes should be added from the left.



If we number the nodes of the complete binary tree from left to right level by level then the node k has its left child at 2k position and right child at 2k+1 position.

Binary Tree Representation in memory

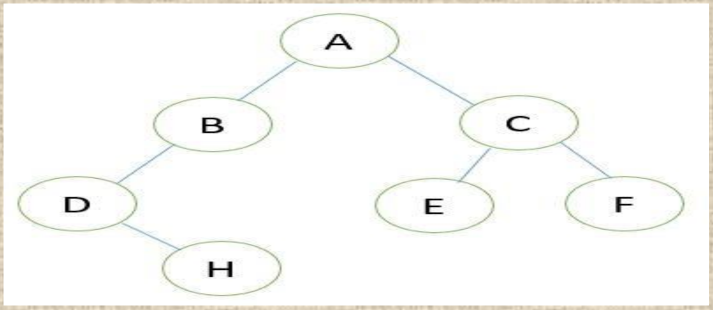
A tree must represent a hierarchical relationship between parent node and child node.
Let T be a Binary Tree. There are two ways of representing T in the memory as follow

1. Sequential or Linear Representation of BinaryTree.
2. Link Representation of Binary Tree.

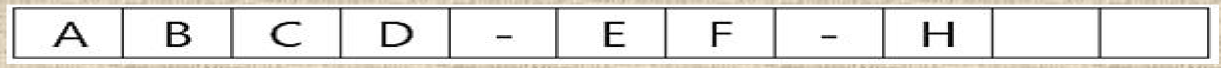
1.Sequential or Linear Representation of BinaryTree.

Let us consider that we have a tree T. let our tree T is a binary tree that us complete binary tree. Then there is an efficient way of representing T in the memory called the sequential representation or array representation of T. This representation uses only a linear array TREE as follows:

1. The root Node N of T is stored in TREE[1].
2. If a node occupies TREE[k] then its left child is stored in TREE[2*k] and its right child is stored into TREE [2 * k + 1].



Its sequential representation is as follow:



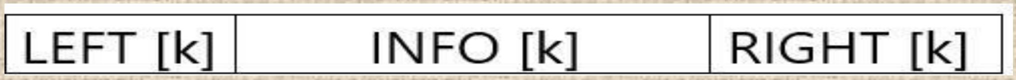
A	B	C	D	-	E	F	-	H		

2.Linked Representation of Binary Tree

Consider a Binary Tree T. Twill be maintained in memory by means of a linked list representation which uses three parallel arrays; INFO, LEFT, and RIGHT pointer
Variable ROOT as follows. In Binary Tree each node N of T will correspond to a location k such that

1. LEFT[k] contains the location of the left child of nodeN.
2. INFO[k]contains the data at the node N.
3. RIGHT[k]contains the location of right child of node N.

Representation of a node:

In this 

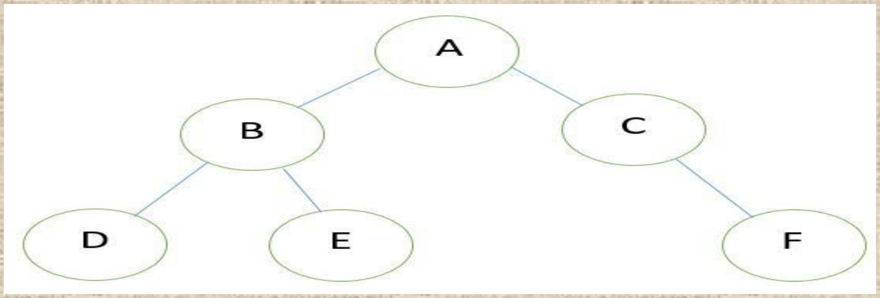
LEFT [k]	INFO [k]	RIGHT [k]

Representation of binary tree root will contain the location of the root R of T. If anyone of the subtree is empty, then the corresponding pointer will contain the null value if the tree T itself is empty, the ROOT will contain the null value.

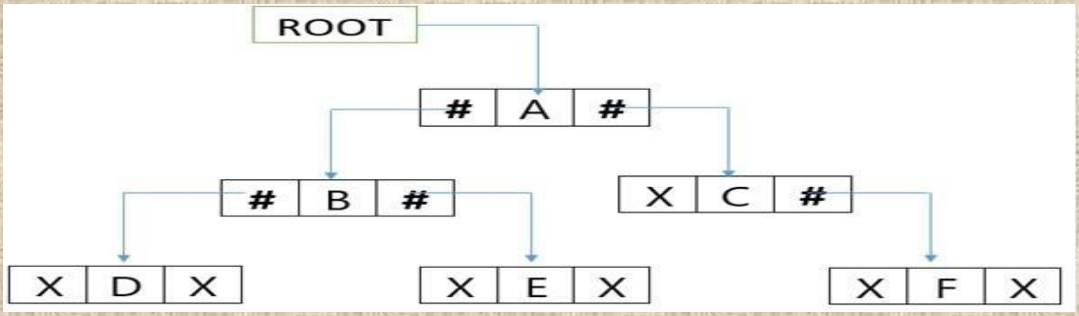
Example

Consider the binarytreeTinthe figure. A schematic diagram of the linked list representation of T appears in the following figure. Observe that each node is pictured with its three fields, and that the empty subtree is pictured by using x for null entries.

BinaryTree



Linked Representation of the Binary Tree:



BinaryTree Traversal

Traversal is a process to visit all the nodes of a tree and may print their values too.Because,all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree.

There are three ways which we use to traverse a tree−

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

1. In-order Traversal

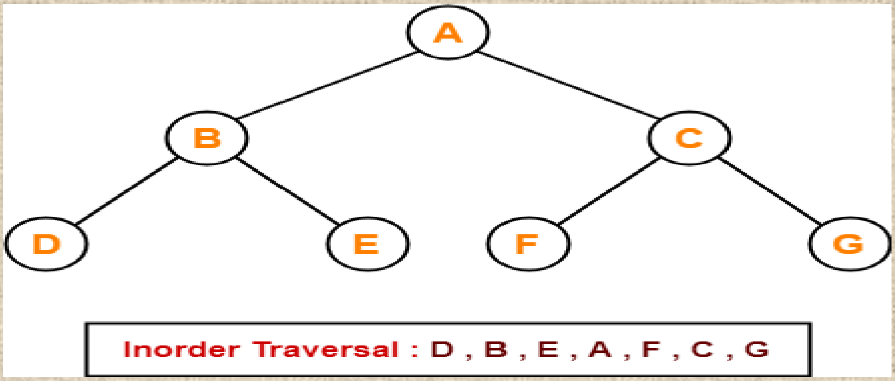
This is also known asymmetric order.

This traversal is referred as LNR.

Algorithm-

1. First, visit all the nodes in the left subtree
2. Then the root node

3. Visit all the nodes in the right subtree
   Left→ Root→Right



Application-

In order traversal is used to get infix expression of an expression tree.

2. PreorderTraversal-

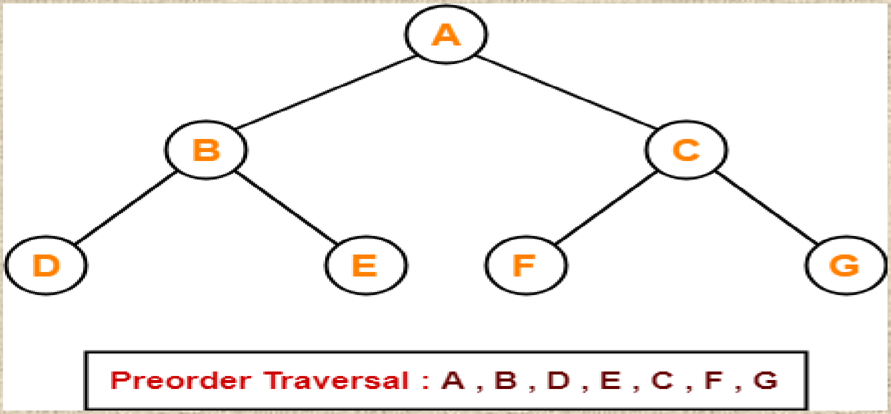
This is also known as depth fistorder.

This traversal is referred as NLR.

Algorithm-

1. Visit the root
2. Traverse the left subtree i.e. call Preorder(leftsubtree)
3. Traverse the right subtreei.e. call Preorder(right subtree)

Root → Left → Right



Applications-

Preorder traversal is used to get prefix expression of an expression tree.Preorder traversal is used to create a copy of the tree.

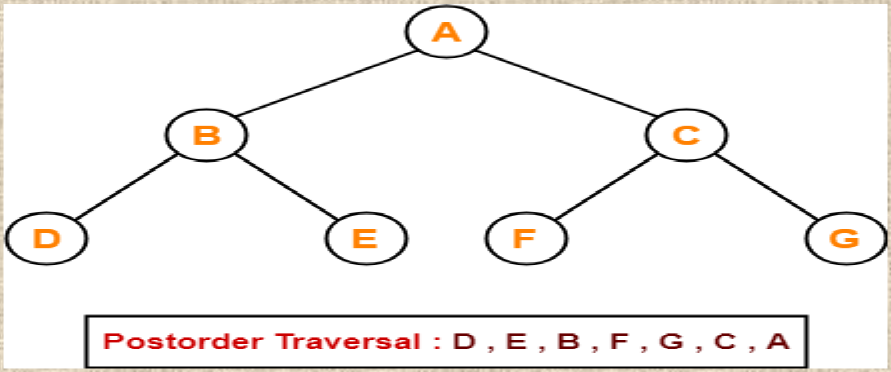
3. Postorder Traversal-

This traversal is referred as LRN.

Algorithm-

1. Traverse the left subtreei.e. call Postorder(leftsubtree)
2. Traverse the right subtreei.e. call Postorder(rightsubtree)
3. Visit the root

Left→Right→Root



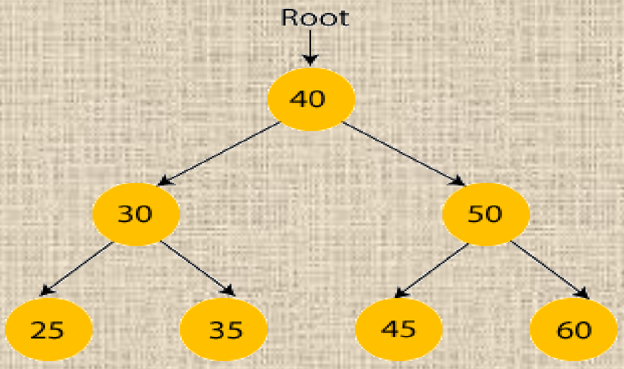
Applications-

- Post order traversal is used to get postfix expression of an expression tree.
- Postorder traversal is used to delete the tree.
- This is because it deletes the children first and then it deletes the parent.

BinarySearchtree

A binary tree T is termed as Binary Search Tree if each noden of T satisfies thefollowing property.

- The value of n is greater than the value of all nodes in its left subtree.
- The value of n is less than the value of all nodes in its right subtree.



Advantages of Binarysearchtree

- Searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element.
- As compared to array and linked lists, insertion and deletion operations are faster inBST.

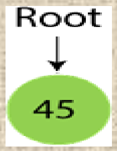
Example of creating a binary search tree

Suppose the data elements are-45,15,79,90,10,55,12,20,50

- First,we have to insert 45 into the tree as the root of the tree.
- Then, read the next element; if it is smaller than the root node, insert it as the root of the left subtree, and move to the next element.
- Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

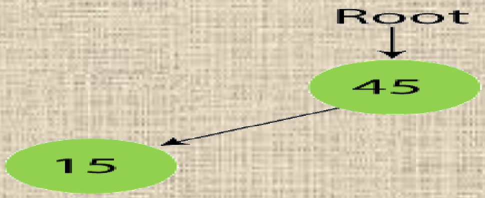
Now, let's see the process of creating the Binary search tree using the given data element. The process of creating the BST is shown below -

Step1-Insert 45.



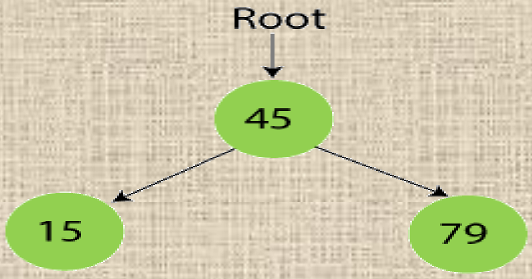
Step 2-Insert15.

As15is smaller than 45,so insert it as the root node of the leftsubtree



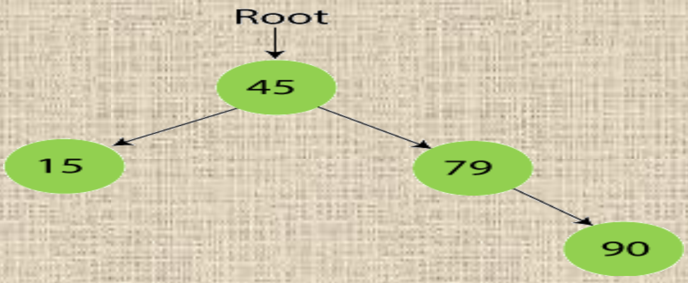
Step3-Insert 79.

As 79 is greater than 45,so insert it as the root node of the right subtree.



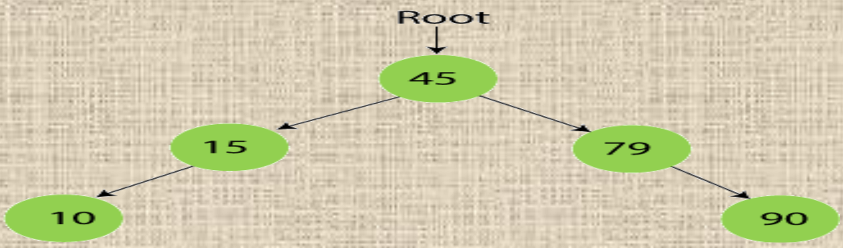
Step4-Insert 90.

90 is greater than 45and79, so it will be inserted as the right subtree of 79**.**



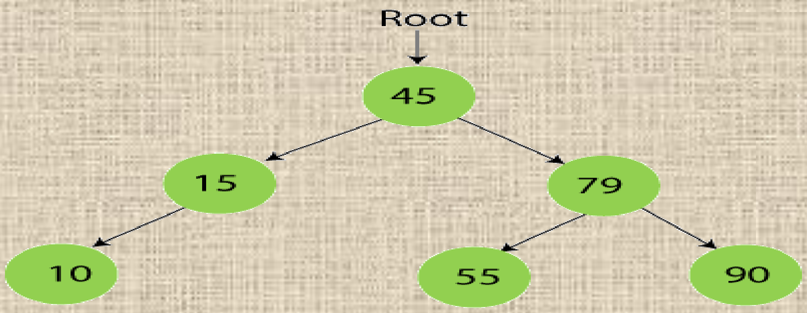
Step5-Insert 10.

10 is smaller than 45 and15, so it will be inserted as a leftsubtree of 15.



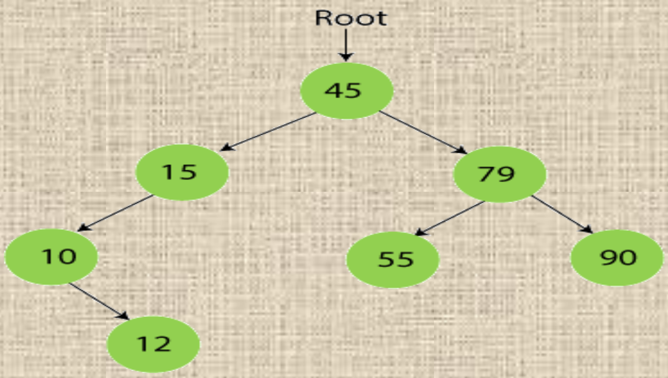
Step6-Insert 55.

55 is larger than 45andsmaller than79, so it will be inserted as the left subtree of 79.



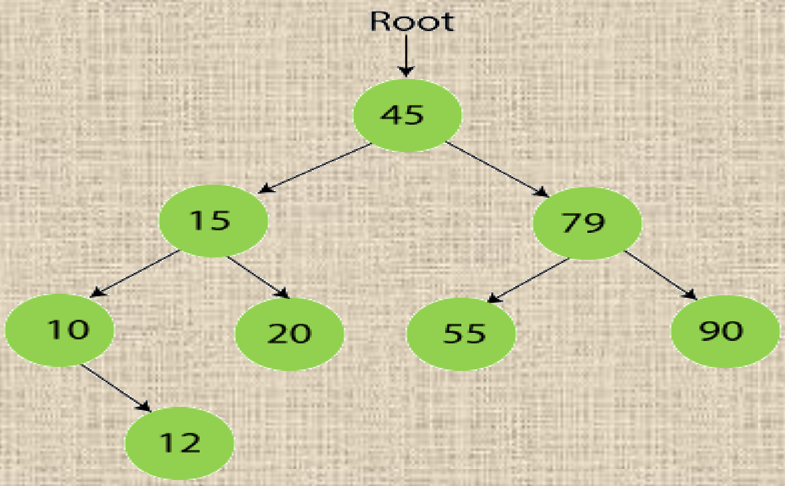
Step7-Insert 12.

12 is smaller than45and15but greater than10,so it will be inserted as the right subtree of 10.



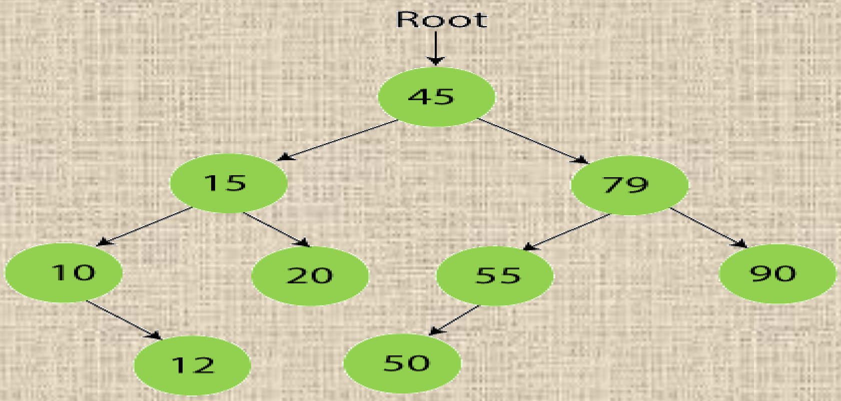
Step8-Insert 20.

20 is smaller than 45 but greater than 15, so it will be inserted as the right subtree of 15.



Step9-Insert 50.

50 is greater than 45 but smaller than 79 and 55. So ,it will be inserted as a left subtree of 55.



Now, the creation of binary search tree is completed.

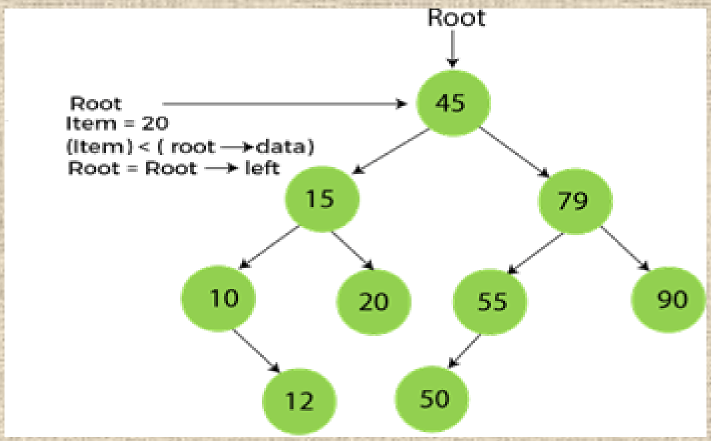
<u>Searching in Binary search tree</u>

Searching means to find or locate a specific element or node in a data structure. In Binary search tree, searching a node is easy because elements in BST are stored in a specific order. The steps of searching a node in Binary Search tree are listed as follows –

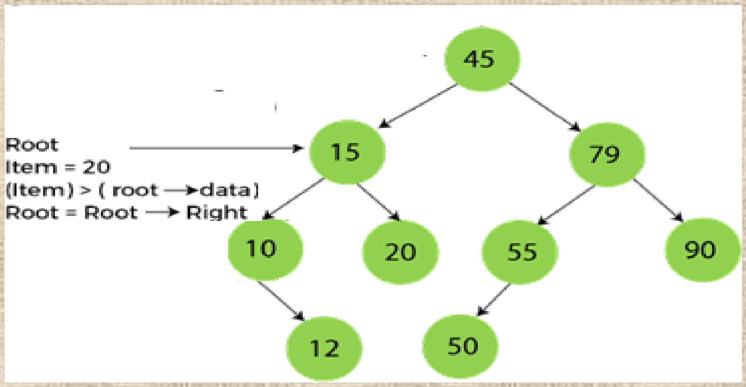
1. First, compare the element to be searched with the root element of the tree.
2. If root is matched with the target element, then return the node's location.
3. If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.
4. If it is larger than the root element, then move to the right subtree.
5. Repeat the above procedure recursively until the match is found.
6. If the element is not found or not present in the tree,then return NULL.

Now, let's understand the searching in binary tree using an example. We are taking the binary search tree formed above. Suppose we have to find node 20 from the below tree.

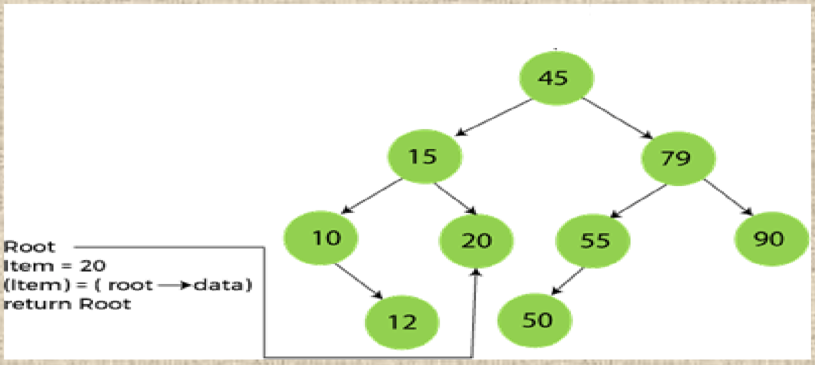
Step1:



Step2:



Step3:



Now, let's see the algorithm to search an element in the Binary search tree.
Algorithm to search an element in Binary search tree

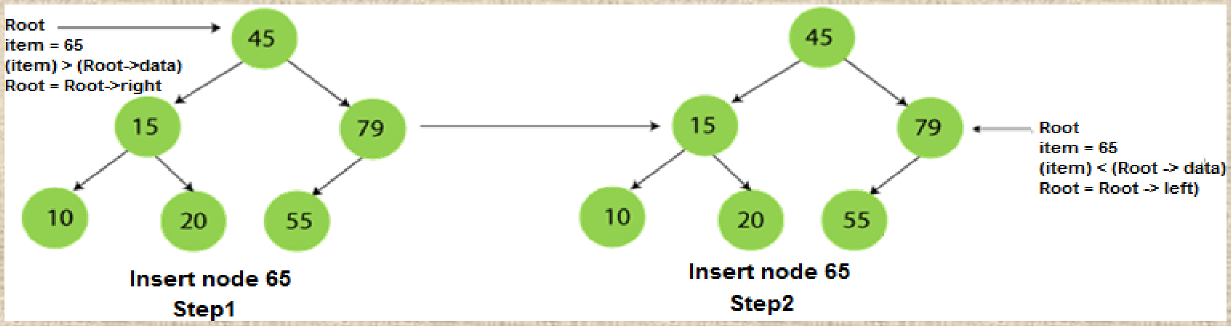
```
1. Search(root,item)
2. Step1-if(item=root→data)or(root=NULL)
3. Return root
4. Else if(item<root→ data)
5. return Search(root→left,item)
6. else
7. return Search(root→right,item)
8. END if
9. Step2-END
```

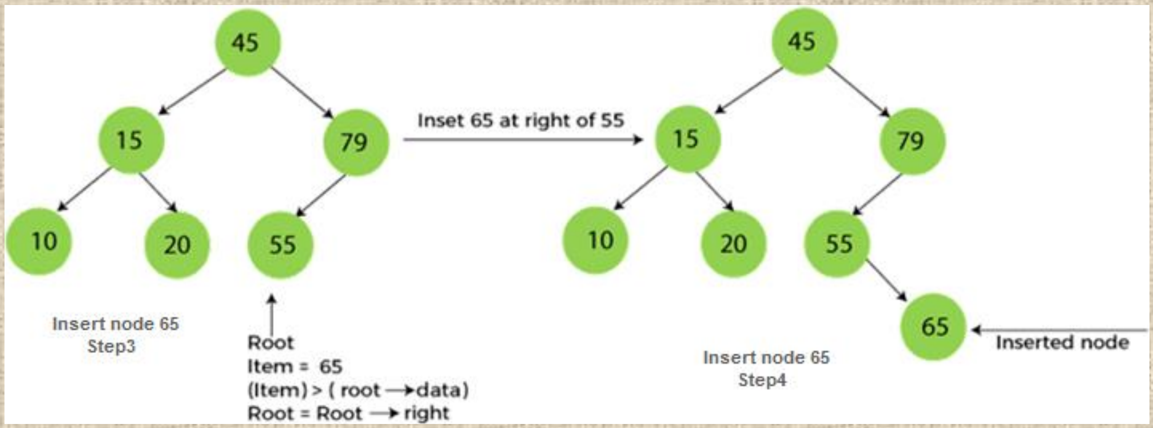
<u>Insertion in Binary Search tree</u>

A new key in BST is always inserted at the leaf.

To insert an element in BST, we have to start searching from the root node; if the node to be inserted is less than the root node, then search for an empty location in the left subtree. Else, search for the empty location in the right subtree and insert the data.

Now, let's see the process of inserting a node into BST using an example.





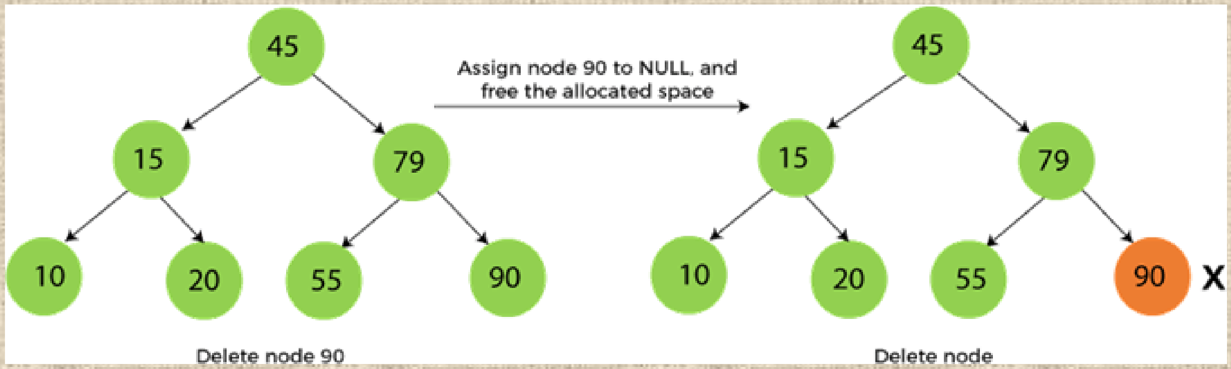
DeletioninBinarySearchtree

In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated. To delete a node from BST, there are three possible situations occur -

- The node to be deleted is the leaf node,or,
- The node to be deleted has only one child ,and,
- The node to be deleted has two children

1. When the node to be deleted is the leaf node

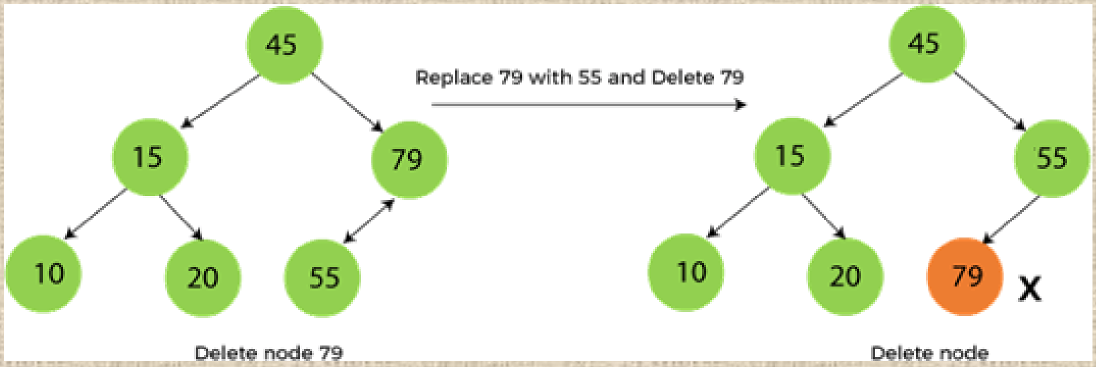
- It is the simplest case to delete a node in BST. Here, we have to replace the leaf node with NULL and simply free the allocated space.
- In below image, suppose we have to delete node 90,as the node to be deleted is a leaf node, so it will be replaced with NULL, and the allocated space will free.



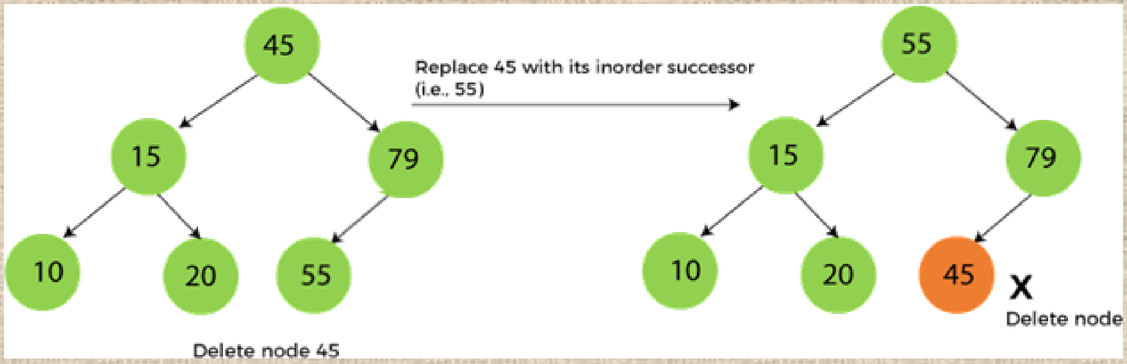
2. When the node to be deleted has only one child

In this case, we have to replace the target node with its child, and then delete the child node. It means that after replacing the target node with its child node, the child node will now contain the value to be deleted. So, we simply have to replace the child node with NULL and free up the allocated space.

In the below image, suppose we have to delete the node 79, as the node to be deleted has only one child, so it will be replaced with its child 55.
So, there placed node79 will now be a leaf node that can be easily deleted.



3. When the node to be deleted has two children
If the node has 2 children ,the strategy is to place the data of this node with the smallest data of the right subtree and recursively delete that node.
In the below image, suppose we have to delete node 45 that is the root node, as the node to be deleted has two children, so it will be replaced with its inorder successor. Now, node 45 will beat the leaf of the tree so that it can be deleted easily.

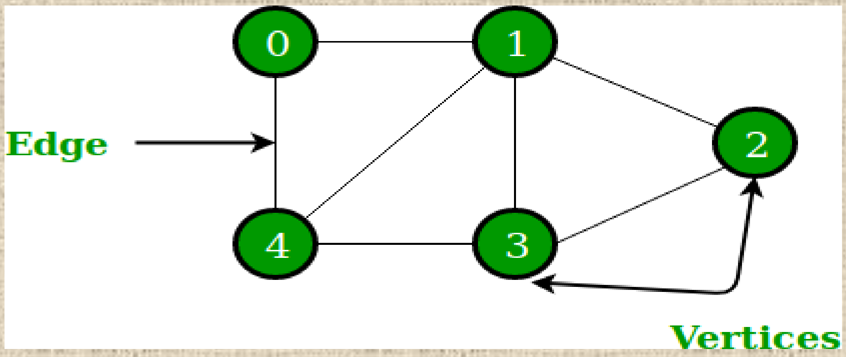


Chapter-7

GRAPH

A graph is a non-linear kind of data structure made up of nodes or vertices and edges.

Formally, a graph is a pair of sets **(V, E)**, where **V**is the set of vertices and **E** is the set of edges, connecting the pairs of vertices.



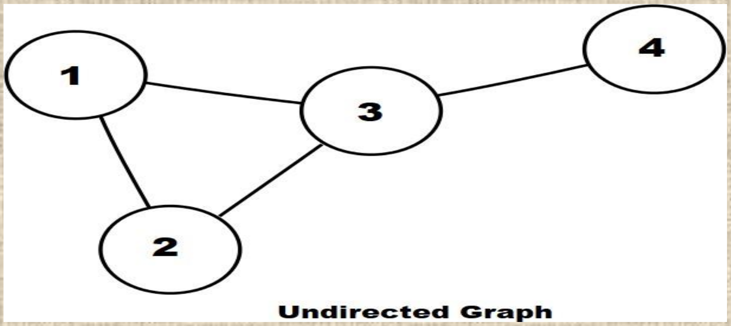
In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

Applications

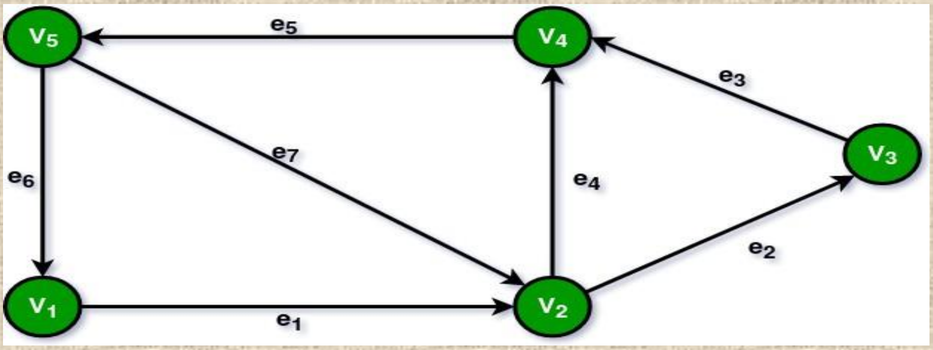
Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(ornode). Each node is a structure and contains information like person id, name, gender etc.

GraphTerminologies

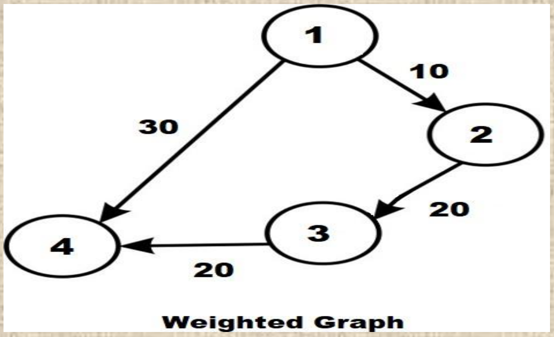
1. Undirected: A graph in which all the edges are bi-directional.The edges do not point in a specific direction.



2. Directed Graph orDigraph: AgraphG=(V, E) with a mapping f such that everyedge maps onto some ordered pair of vertices (Vi, Vj) is called Digraph. It is also called *Directed Graph*. Ordered pair (Vi, Vj) means an edge between Vi and Vj with an arrow directed from Vi to Vj.

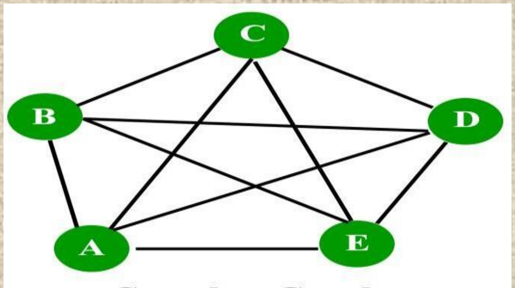


3. Weighted Graph:A graph that has a value associated with every edge. The values corresponding to the edges are called weights. A value in a weighted graph can represent quantities such as cost, distance, and time, depending on the graph. Weighted graphs are typically used in modeling computer networks.



4. Unweighted Graph:A graph in which there is no value or weight associated with the edge. All the graphs are unweighted by default unless there is a value associated.

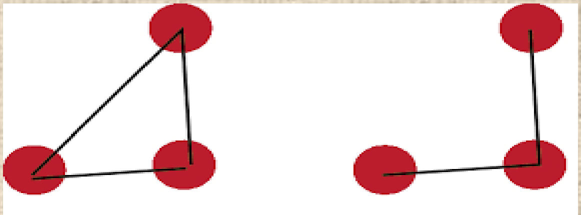
5. <u>Complete Graph:</u>A simple graph with n vertices is called a complete graph if the degree of each vertex is n-1, that is, one vertex is attach with n-1 edges. A complete graph is also called Full Graph.



Complete Graph

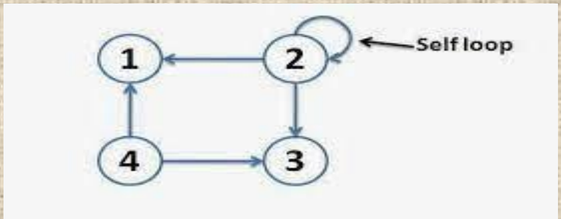
6. <u>Cyclic Graph:</u> For a graph to be called acyclic graph, it should consist of at least one cycle. If a graph has a minimum of one cycle present, it is called acyclic graph.

7. <u>Acyclic Graph:</u> A graph is called an acyclic graph if zero cycles are present, and an acyclic graph is the complete opposite of a cyclic graph.



Cyclic Acyclic

<u>8.Vertex</u>:Every individual data element is called a vertex or a node.
<u>9.Edge</u>It is a connecting link between two nodes or vertices.
10. <u>Degree</u>: The total number of edges connected to a vertex in a graph.
Indegree: The total number of incoming edges connected to a vertex.
Outdegree The total number of outgoing edges connected to a vertex.
11. <u>Self-loop</u>: If there is an edge whose stat and end vertices are same i.e.(vi,vi)is an edge ,then it is called self loop.



Graph with self loop

12. **Adjacent Nodes:** Two nodes are called adjacent if they are connected through an edge.
13. **Path:** sequence of vertices in which each pair of successive vertices is connected by an edge
14. **IsolatedVertex:** A vertex with degree zero is called an isolated vertex.

Example:

Here, the vertex 'a' and vertex'b' has a no connectivity between each other and also to any other vertices. So the degree of both the vertices 'a' and 'b' are zero. These are also called as isolated vertices.

RepresentationofGraph

The graph can be represented as matrices.

Adjacency Matrix

Adjacency Matrix is used to represent a graph. We can represent directed as well as undirected graphs using adjacency matrices..

If a graph has n number of vertices, then the adjacency matrix of that graph is n x n, and each entry of the matrix represents the number of edges from one vertex to another.

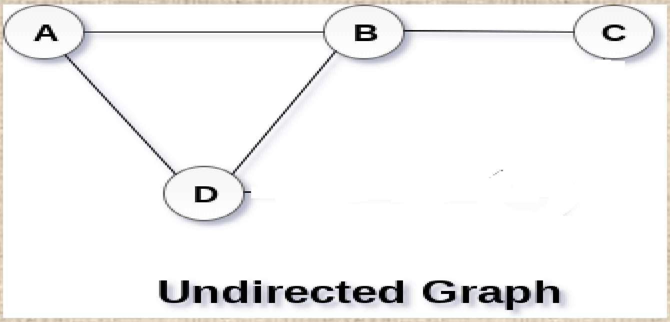
Adjacency Matrix Representation for undirectedgraph

If an Undirected Graph G consists of n vertices then the adjacency matrix of a graph is n x n matrix A = [aij] and defined by -

aij=1{if there is a path exists fromVitoVj}

aij = 0 {Otherwise}

In an undirected graph,if there is an edge exists between Vertex AandVertex B,then the vertices can be transferred from A to B as well as B to A.

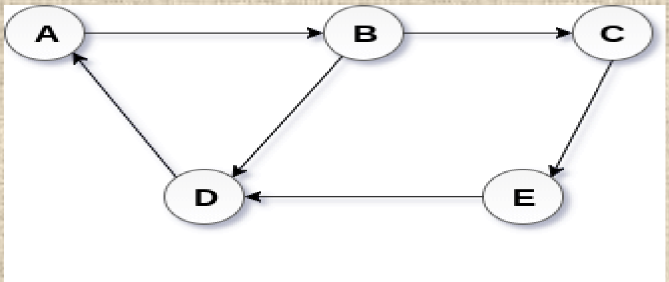


The adjacency matrix of the above graph will be-

	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	0	0

Adjacency matrix representation for a directedgraph

In a directedgraph, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertexB. Node A is called the initial node, while node B is called the terminal node.



Directed Graph

The adjacency matrix of the above graph will be-

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	1	0
C	0	0	0	0	1
D	1	0	0	0	0
E	0	0	0	1	0

Properties of the adjacency matrix

Some of the properties of the adjacency matrix are listed as follows:

- An adjacency matrix is a matrix that contains rows and columns used to represent a simple labeled graph with thenumbers 0and1 in the positionof(V_I, V_j),according to the condition of whether or not the two V_i and $V_j$are adjacent.
- For a directed graph, if there is an edge exists between vertex ior V_i to Vertex jor V_j, then the value of $A[V_i][V_j] = 1$, otherwise the value will be 0.
- For an undirected graph, if there is an edge that exists between vertex I or V_i to Vertex j or V_j, then the value of $A[V_i][V_j] = 1$ and $A[V_j][V_i] = 1$, otherwise the value will be 0.

Chapter-8

LINKED REPRESENTATI ON OF GRAPH

Graph is a mathematical structure and finds its application in many areas, where the problem is to be solved by computers. The problems related to graph G must be represented in computer memory using any suitable data structure to solve the same. There are two standard ways of maintaining (representation)a graph Ginthememory of a computer.

1. Sequential representation of a graph using adjacent(adjacencymatrix)
2. Linked representation of a graph using linkedlist(adjacencylist)

1. ADJACENCY MATRIX REPRESENTATION

The SA of a graph G=(V,E)with n vertices, is an n×n matrix. In this section let us see how a directed graph can be represented using adjacency matrix. Considered a directed graph in Fig. 9.12 where all the vertices are numbered, (1, 2, 3, 4etc.)

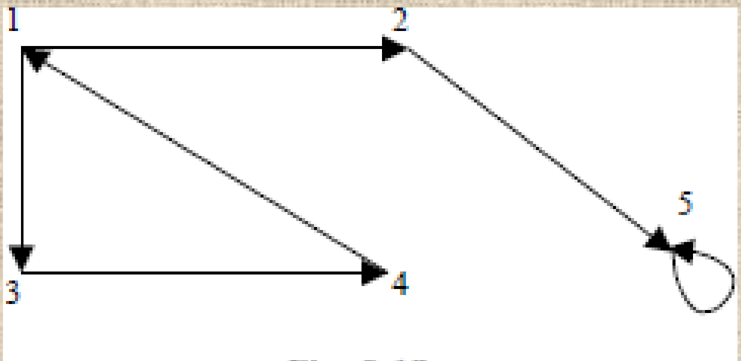
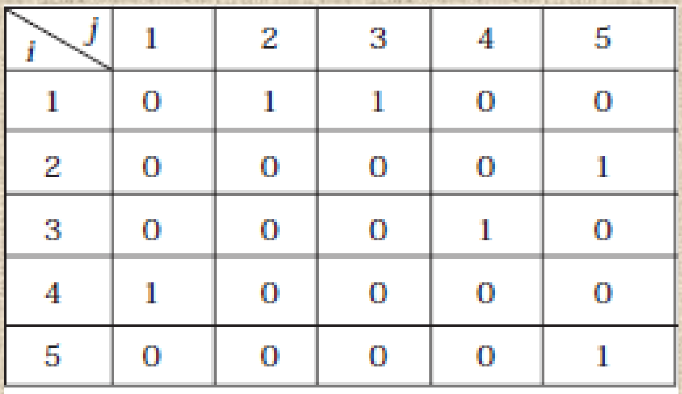


Fig. 9.12

The adjacency matrix A of a directed graphG=(V,E)canberepresented(inFig 9.13) with the following conditions

A_{ij}=1{if there is an edge from V_i to V_j or if the edge(i,j)ismemberofE.} A_{ij} = 0 {if there is no edge from V_i to V_j}



i \ j	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	0	1
3	0	0	0	1	0
4	1	0	0	0	0
5	0	0	0	0	1

Fig. 9.13

Wehaveseenhowadirectedgraphcanberepresentedinadjacencymatrix.Now let us discuss how an undirected graph can be represented using adjacency matrix. ConsideredanundirectedgraphinFig.9.14

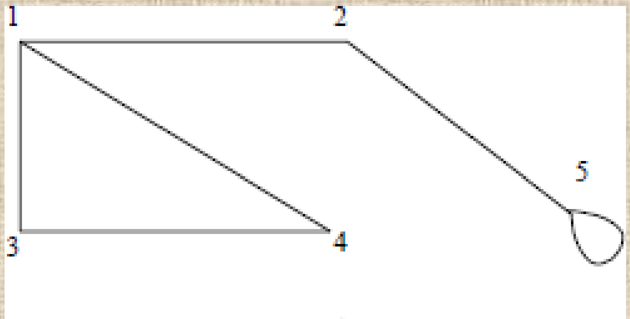
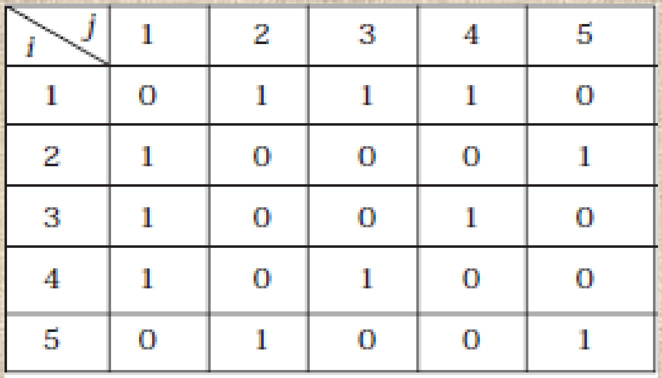


Fig. 9.14

TheadjacencymatrixAofanundirectedgraphG=(V,E)canberepresented(inFig 9.15) with the following conditions

A_{ij} = 1 {if there is an edge from V_i to V_j or if the edge (i, j) is member of E} A_{ij}=0{if there

is no edge from$V_i$to$V_j$or the edgei,j,is not a member of E}



i \ j	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	0	1
3	1	0	0	1	0
4	1	0	1	0	0
5	0	1	0	0	1

Fig. 9.15

To represent a weighted graph using adjacency matrix, weight of the edge (i, j) is simply stored as the entry in Ith row and jth column of the adjacency matrix.There

Are some cases where zero canal so be the possible weight of the edge, the n we have to store some sentinel value for non-existent edge, which can be a negative value; since the weight of the edge is always a positive number. Consider a weighted graph, Fig. 9.16

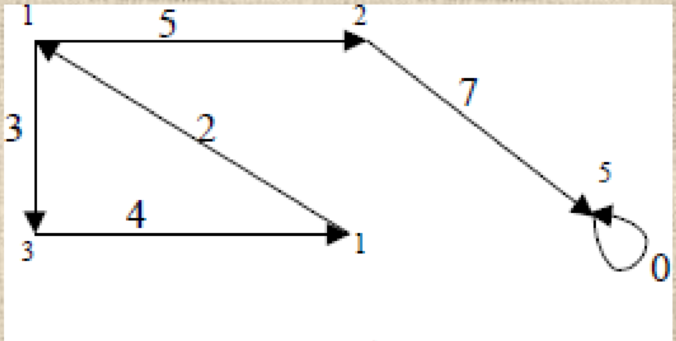
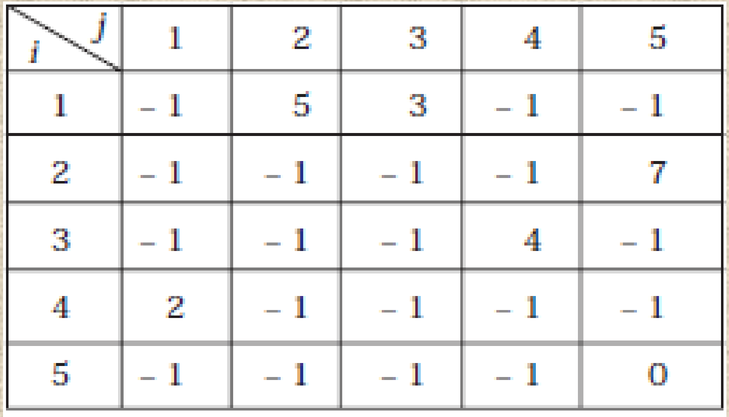


Fig. 9.16

The adjacency matrix A for a directed weighted graph G = (V, E, W_e) can be represented (in Fig. 9.17) as

A_{ij}=W_{ij}{if there is an edge fromV_i to$V_j$the n represent its weightW_{ij}.} A_{ij} = – 1 { if there

is no edge from V_i to V_j}



j / i	1	2	3	4	5
1	– 1	5	3	– 1	– 1
2	– 1	– 1	– 1	– 1	7
3	– 1	– 1	– 1	4	– 1
4	2	– 1	– 1	– 1	– 1
5	– 1	– 1	– 1	– 1	0

Fig. 9.17

In this representation, n^2 memory location is required to represent a graph with n vertices. The adjacency matrix is a simple way to represent a graph, but it has two disadvantages.

1. It takes n^2space to represent a graph with n vertices,even for as parse graph.

2. It takesO(n^2)time to solve the graph problem

2. LINKEDLIST REPRESENTATION

Inthisrepresentation(alsocalledadjacencylistrepresentation),westoreagraph as a linked structure. First we store all the vertices of the graph in a list and then each adjacent vertices will be represented using linked list node. Here terminal vertex of an edgeisstoredinastructurenodeandlinkedtoacorrespondinginitialvertexinthelist. Consider a directed graph in Fig. 9.12, it can be represented using linked list as Fig. 9.18.

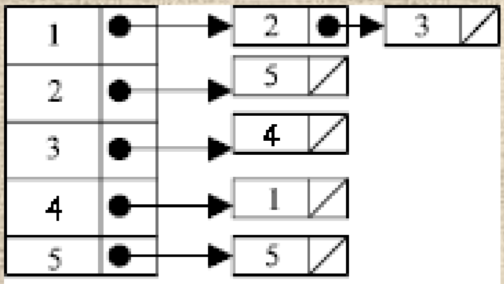


Fig. 9.18

Weighted graph can be represented using linkedlist by storing the corresponding weight along with the terminal vertex of the edge. Consider a weighted graph in Fig. 9.16, it can be represented using linked list as in Fig. 9.19.

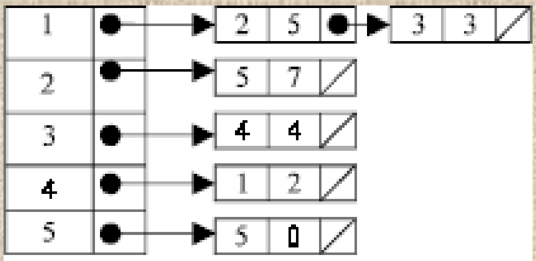


Fig. 9.19

Although the linked list representation requires very less memory as compared to the adjacency matrix, the simplicity of adjacency matrix makes it preferable when graph are reasonably small.

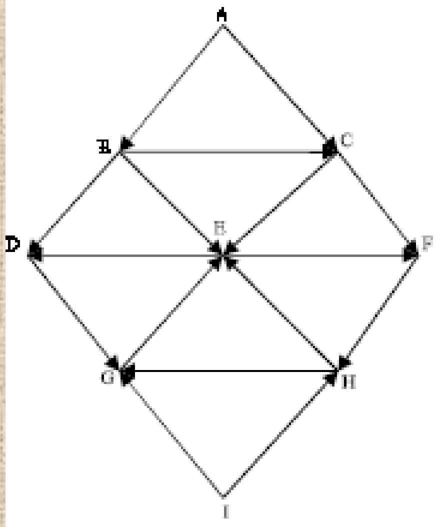
TRAVERSING A GRAPH

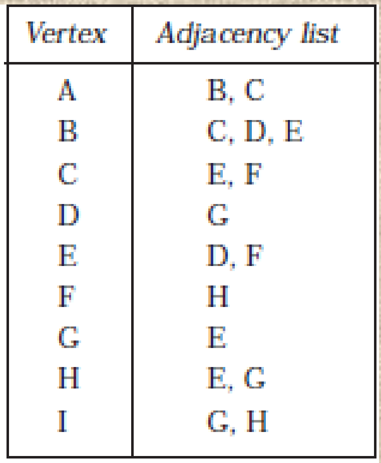
Many application of graph requires a structured system to examine the vertices and edges of a graph G.That is a graph traversal,which means visiting all the nodes of the graph. There are two graph traversal methods.

(a) Breadth First Search(BFS)
(b) Depth First Search(DFS

BREADTH FIRST SEARCH

Give n an input graph G=(V,E)and as our cevertex S,from where the searching starts. The breadth first search systematically traverse the edges of G to explore every vertex that is reachable from S. Then we examine all the vertices neighbor to source vertexS.Then we traverse all the neighbors of the neighbors of source vertex Sand so on.A queue is used to keep track of the progress of traversing the neighbor nodes. BFS can be further discussed with an example. Considering the graph G in Fig. 9.20



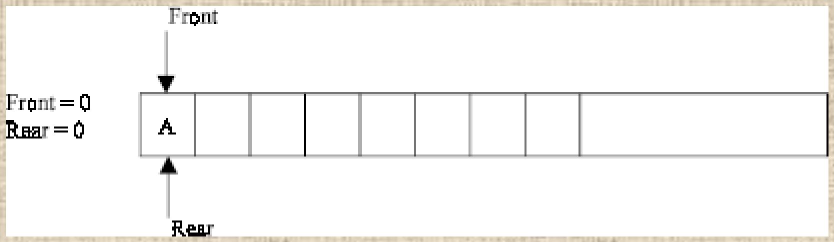


Vertex	*Adjacency list*
A	B, C
B	C, D, E
C	E, F
D	G
E	D, F
F	H
G	E
H	E, G
I	G, H

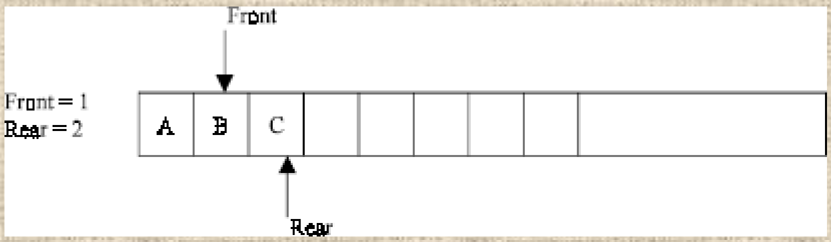
Fig. 9.20

The linked list (or adjacency list) representation of the graph Fig. 9.20 is also shown. Suppose the source vertex is A. Then following steps will illustrate the BFS.

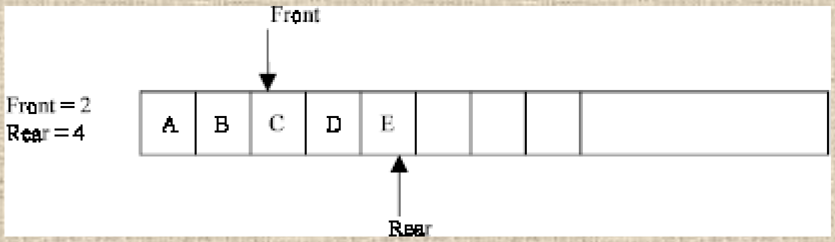
Step1:Initially pushA(the source vertex)tothe queue.



Step 2: Pop (or remove) the front element A from the queue (by incrementing front = front+1)and display it. Then push(or add)the neighboring vertices of A to the queue, (by incrementing Rear = Rear +1) if it is not in queue.

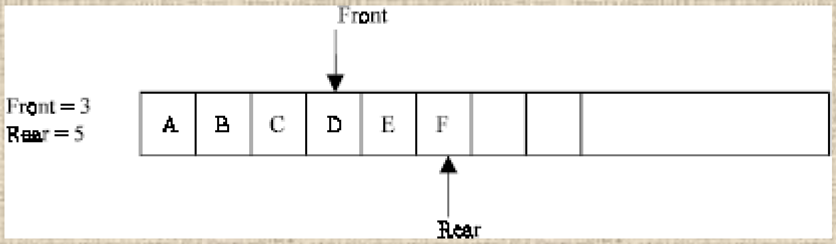


Step3:Pop the front element B from the queue and display it.The n add the neighboring vertices of B to the queue, if it is not in queue.



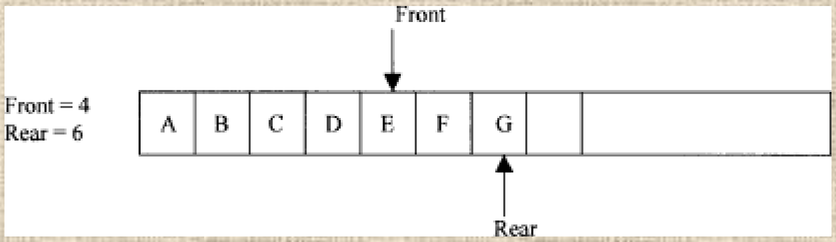
One of the neighboring element C of B is preset in the queue, So C is not added to queue.

Step 4: Remove the front element C and display it. Add the neighboring vertices of C, if it is not present in queue.

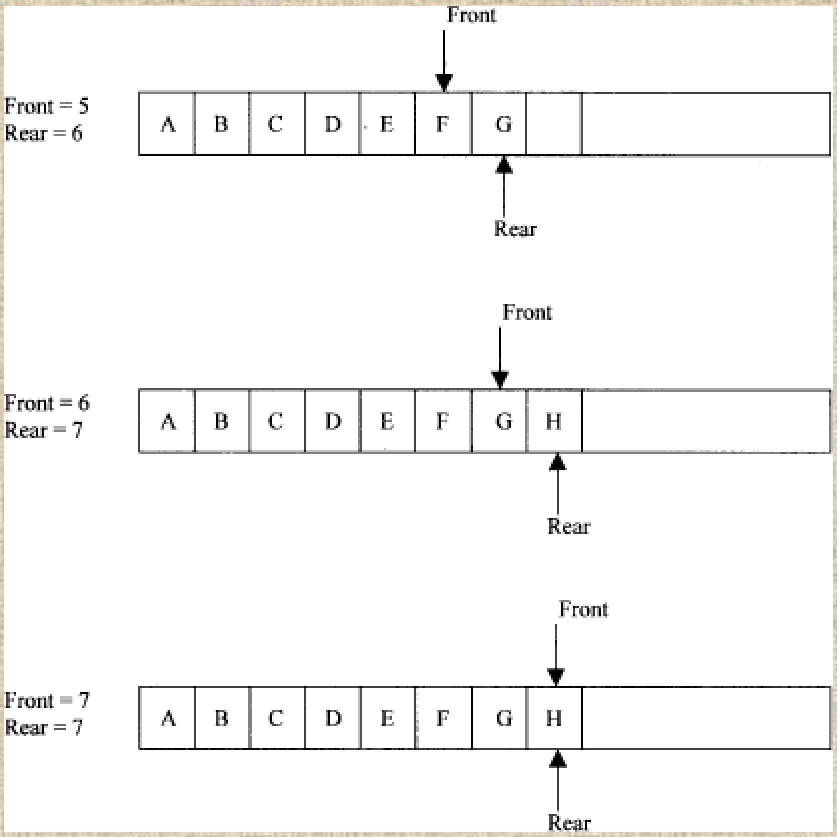


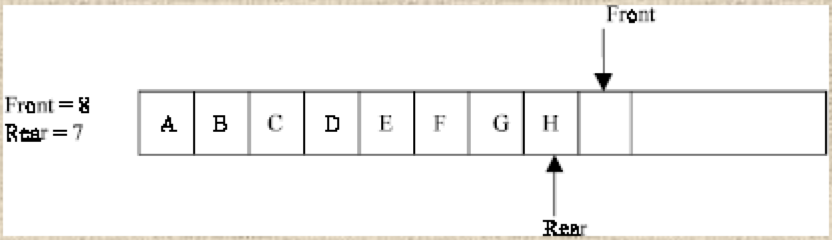
One of the neighboring elements E of C is present in the queue. So E is not added.

Step5: Remove the front element D,and add the neighboring vertex if it is not present in queue.



Step 6: again the process is repeated (until front > rear). That is remove the front element E of the queue and add the neighboring vertex if it is not present in queue.





SoA,B,C,D,E, F,G,His the BFS traversal of the graph in Fig.9.20

ALGORITHM

1. Input the vertices of the graph and its edgesG=(V,E)
2. Input the source vertex and assign it to the variableS.
3. Add or push the source vertex to the queue.
4. Repeat the steps5 and 6until the queue is empty(i.e.,front>rear)
5. Pop the front element of the queue and display it as visited.
6. Push the vertices,which is neighbor to just,popped element, if it is not in the queue and displayed (i.e., not visited).
7. Exit.

(b) DEPTH FIRST SEARCH

The depthfirstsearch(DFS), as its name suggest,is to search deeper in the graph, whenever possible.Give n an input graphG=(V,E)and as our c evertexS,from where the searching starts. First we visit the starting node. Then we travel through each node along a path, which begins at S. That is we visit a neighbor vertex of S and again a neighbor of a neighbor of S, and so on. The implementation of BFS is almost same except a stack is used instead of the queue. DFS can be further discussed with an example.Consider the graph inFig.9.20and its linkedlist representation.Suppose the source vertex is I. The following steps will illustrate the DFS

Step1:InitiallypushIontothestack.

<u>STACK: I</u>

<u>DISPLAY:</u>

Step2:Popanddisplaythetopelement,andthenpushalltheneighborsofpopped element (i.e., I) onto the stack, if it is not visited (or displayed or not in the stack).

<u>STACK:G,H DISPLAY: I</u>

Step 3: Pop and displaythe top element and then push all the neighbors of popped the element (i.e., H) onto top of the stack, if it is not visited.

<u>STACK: G, E DISPLAY:I,H</u>

The popped element H has two neighbors Eand G. G is already visited, means G is either in the stack or displayed. Here G is in the stack. So only E is pushed onto the top of the stack.

Step4: Pop and display the to p element of the stack.Push all the neighbors of the popped element on to the stack, if it is not visited.

<u>STACK: G, D, F DISPLAY:I,H,E</u>

Step5: Pop and display the to p element of the stack.Push all the neighbors of the popped element onto the stack, if it is not visited.

<u>STACK: G, D DISPLAY:I,H,E,F</u>

The popped element(orvertex)F has neighbor(s)H, which is already visited. Then His displayed, and will not be pushed again on to the stack.

Step 6: The process is repeated as follows.

STACK:G

DISPLAY:I,H,E,F,D

STACK: //nowthestackisempty

DISPLAY: I, H, E, F, D, G

SoI,H,E,F,D,G is the DFS traversal of graph Fig9:2 0 from the source vertexI.

ALGORITHM

1. Input the vertices and edges of the graphG=(V,E).
2. Input the source vertex and assign it to the variableS.
3. Push the source vertex to the stack.
4. Repeat the steps 5 and 6 until the stack is empty.
5. Pop the top element of the stack and display it.
6. Push the vertices which is neighbor to just popped element, if it is not in the queue and displayed (i.e.; not visited).
7. Exit.