MAA OMWATI DEGREE COLLEGE HASSANPUR (PALWAL)

Notes

BCA 3rd Sem

Data Structure -I

BCA – 202 : DATA STRUCTURES – I

External Marks: 80
Internal Marks: 20

Time: 3 hours

Note: Examiner will be required to set NINE questions in all. Question Number 1 will consist of total 8 parts (short-answer type questions) covering the entire syllabus and will carry 16 marks. In addition to the compulsory question there will be four units i.e. Unit-I to Unit-IV. Examiner will set two questions from each Unit of the syllabus and each question will carry 16 marks. Student will be required to attempt FIVE questions in all. Question Number 1 will be compulsory. In addition to compulsory question, student will have to attempt four more questions selecting one question from each Unit.

UNIT – I

Introduction: Elementary data organization, Data Structure definition, Data type vs. data structure, Categories of data structures, Data structure operations, Applications of data structures, Algorithms complexity and time-space tradeoff, Big-O notataion.

Strings: Introduction, Storing strings, String operations, Pattern matching algorithms.

UNIT – II

Arrays: Introduction, Linear arrays, Representation of linear array in memory, address calculations, Traversal, Insertions, Deletion in an array, Multidimensional arrays, Parallel arrays, Sparse arrays.

Linked List: Introduction, Array vs. linked list, Representation of linked lists in memory, Traversal, Insertion, Deletion, Searching in a linked list, Header linked list, Circular linked list, Two-way linked list, Threaded lists, Garbage collection, Applications of linked lists.

UNIT – III

Stack: Introduction, Array and linked representation of stacks, Operations on stacks, Applications of stacks: Polish notation, Recursion.

Queues: Introduction, Array and linked representation of queues, Operations on queues, Deques, Priority Queues, Applications of queues.

UNIT – IV

Tree: Introduction, Definition, Representing Binary tree in memory, Traversing binary trees, Traversal algorithms using stacks.

Graph: Introduction, Graph theory terminology, Sequential and linked representation of graphs.

SUGGESTED READINGS

1. Seymour Lipschutz, "Data Structure", Tata-McGraw-Hill
2. Horowitz, Sahni & Anderson-Freed, "Fundamentals of Data Structures in C", Orient Longman.
3. Trembley, J.P. And Sorenson P.G., "An Introduction to Data Structures With Applications", Mcgraw- Hill International Student Edition, New York.
4. Mark Allen Weiss Data Structures and Algorithm Analysis In C, Addison- Wesley, (An Imprint Of Pearson Education), Mexico City.Prentice- Hall Of India Pvt. Ltd., New Delhi.
5. Yedidyah Langsam, Moshe J. Augenstein, and Aaron M. Tenenbaum, "Data Structures Using C", Prentice- Hall of India Pvt. Ltd., New Delhi.

Note: Latest and additional good books may be suggested and added from time to time.

S.No. 068

DELTA Pg No.
Date 6/8/15

Unit 1

Introducation :-

* Datastructure → D.S is a mathematical & logical way to express the data in more efficient manner.

a19b20df+3*

abdf 19203 +*

* :- D.S is also a model to structurised the data with the heep of access function where the access function links to the types of data. e.g. if Array int m[10] is a data & then m is the access function which means, it contains 10 integer value

objectives of data structure:-

① It enables the inherent relationship of data with the real world.

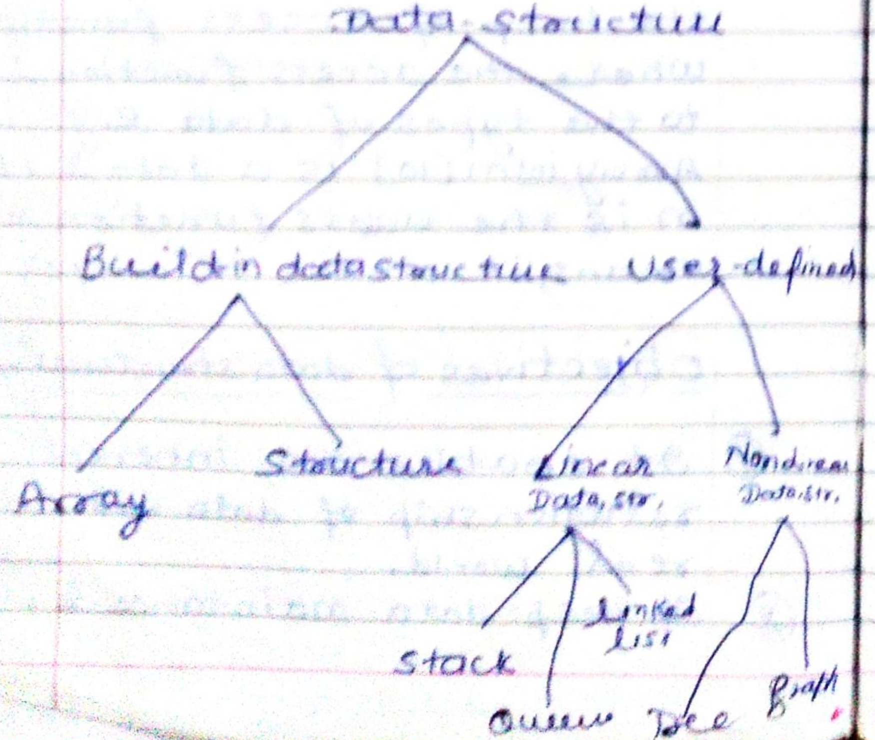
② It heep data maintance & recorded

(3) It enables the data integrity.
(4) It enables the proper storage of data in more efficient manner

⇛ Types of data structure

Data-structure are of two types

(1) Build in data structure.
(2) User-defined data structure



(1) Build in data structure :-

Build in data structure are those data structure which are internally provided by the high level language.
These type of data structure are also formed with the help of primary data-type like int, char, float.

two. Types of Build-in data structure.

(1) Array.
(2) Structure.

(1) Array :-

A finite ordered set of homogenous elements is called an array where
finite mean elements are countable
Ordered set mean all the elements have a position
& homogenous means all the elements of same data type.

e.g:→

array name

int number[10]
datatype size of array

where number is the name of the array

3	9	7	8

[10] means total no. of elem. are 10 & int means all the element are integer type

② Structure→

Structure is a data item which can be use to collect heterogenous element in a single unit is called structure. It is denoted by Struct keyword.

Syntax-

```
Struct structurename
 {
Data~~type~~1   Dataitem1;
Datatype2   Dataitem2;
Datatype3    item3;
 |
 |
 |
Dataitem n   item n;
};
```

Example:

```
Struct student
 {
int  roll.no;
~~name~~ ~~char~~ char  name[10];
int  cont.no;
float  marks;
 };
```

ⓑ Userdefined data structure:

A data-structure is called user-defined data structure which are formed as the requirement of the user. ~~(requirement)~~

It has two type

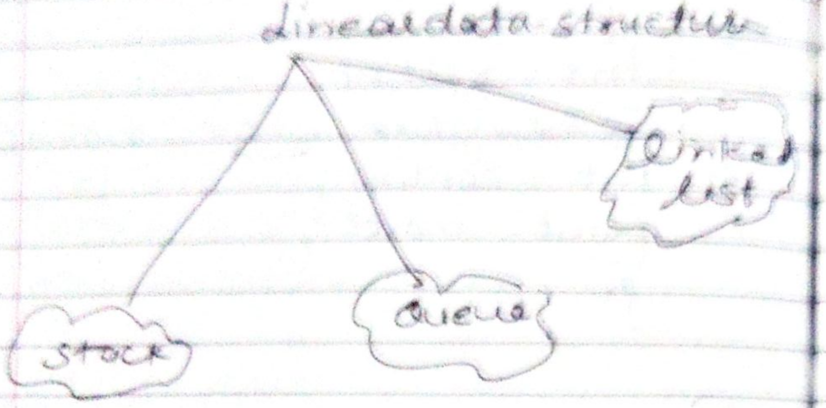
(I) Linear D.S

(II) Non-Linear D.S

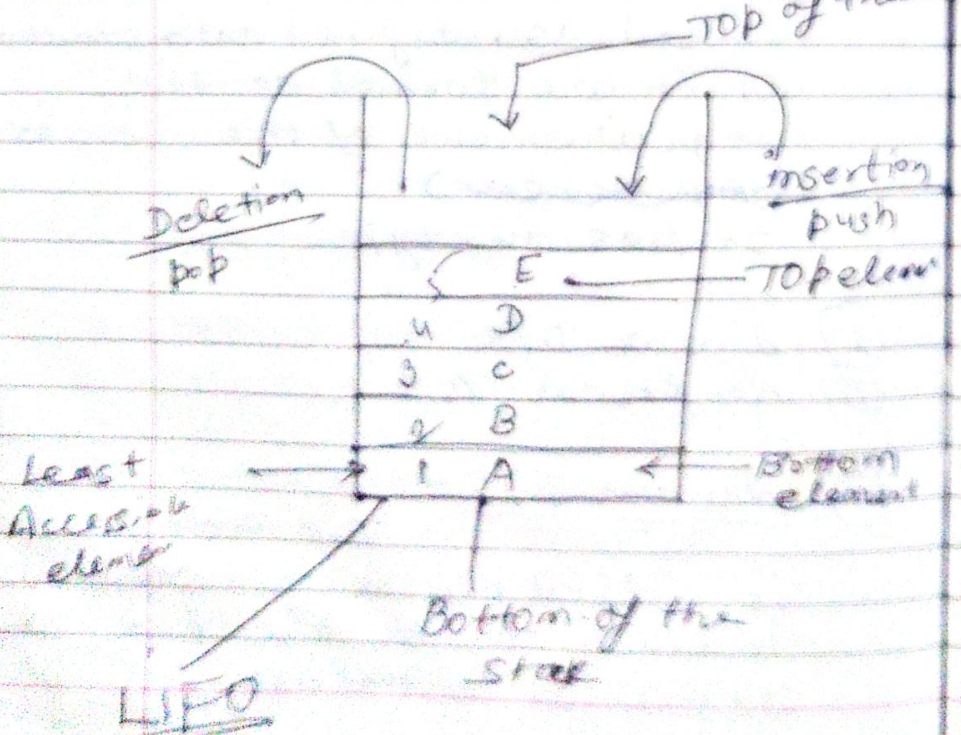
(I) Linear D.S→

The Data structure is called linear data-structure in which all the element are sequence in nature.

Linear data structure are of thee type



① Stack :-



Ex Coin stacker

Ex Coin stacker

* Stack is a linear data structure in which insertion & deletion take place at the same end called top of the stack.

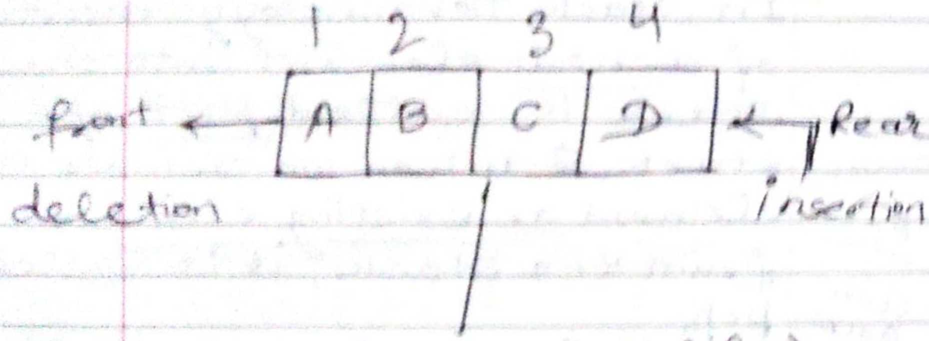
NOEL(S) represent no. of element in the stack. So in above diagram, The NOEL(S) = 5

In stack Terminology, insertion of a new element into the stack is called push of the stack & when we remove the element or existing element from the stack, it is called pop.

So in the stack the bottom element is the least accessible element & the top element of the stack is most accessible element bcz top element is always change due to the push & pop operations.

In stack, The principle of stack is based upon on LIFO (last in first out) procedure i.e the element inserted into the last position must be accessed or removed first.

② Queue → It is also linear data structure in which insertion can be done at rear and & deletion can be done at front end.



Queue (Fifo)

in above example, we have Rear end for insertion, so we can easily insert the elements from A to D & also we can easily deleted from ~~front end~~ from A to D but it can be done only & only Fifo manner.

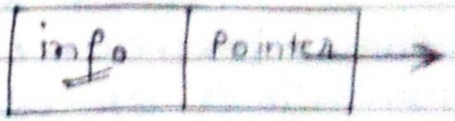
So The principle of Queue is Fifo i.e The element inserted first, delete also first.

Example -

The number of passenger are standing on a ticket counter of Railway station for getting the tickets.

③ Linked List →

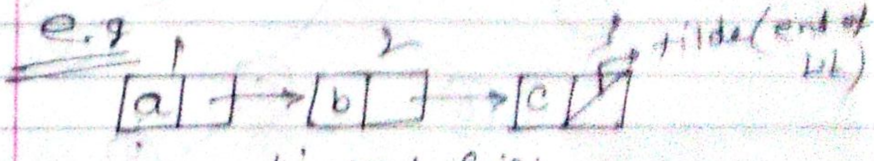
It is also a linear data structure which contain two thing one is information & other is pointer.



Node (linked list)

So It is a collection of information as well as pointer which point to the next node

e.g



Linked list

(b) Non-linear data-structure →

It is a data structure in which all the elements are hierochy or non-sequence in nature.

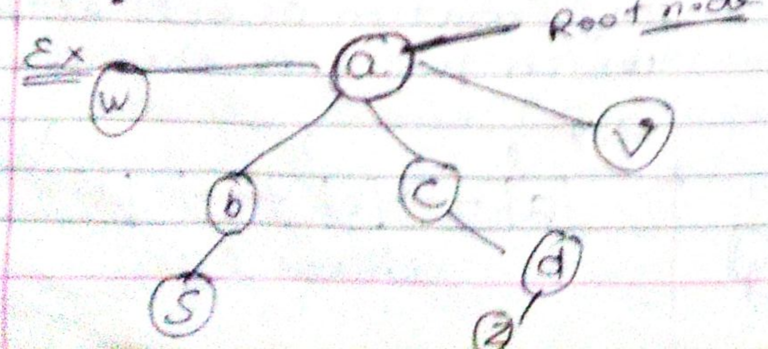
Non-linear data structure are of 2-type

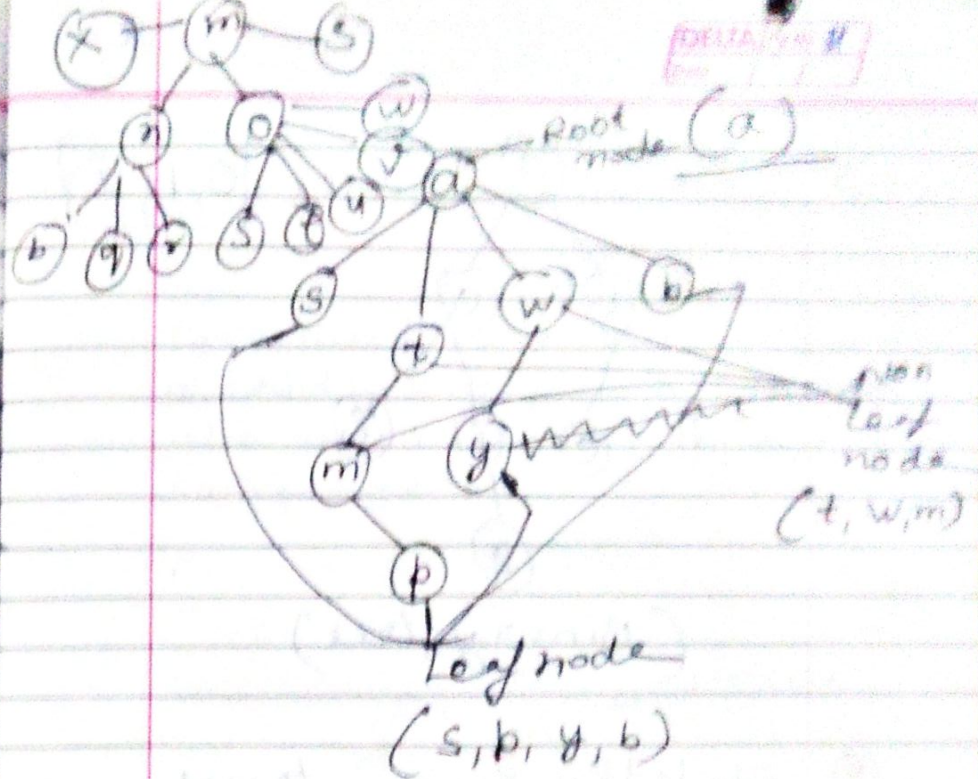
1. Tree
2. Graph

(1) Tree :- (General tree)

It is a non-linear data-structure in which it is contain a special node called root of the tree & remaining nodes are the leaf & non-leaf nodes

Ex

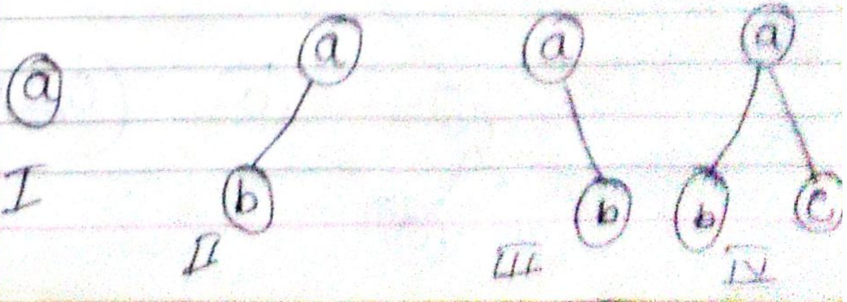


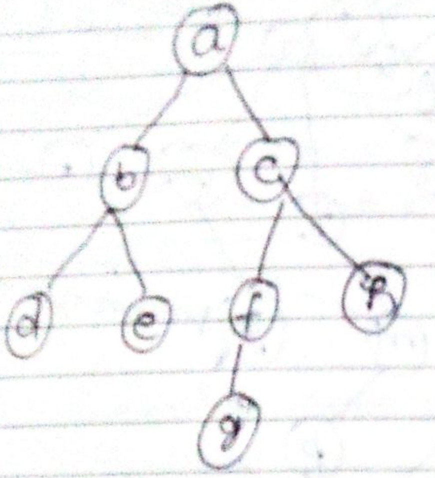


Types of tree :-

(1) Binary tree :-

A tree is said to be binary tree if there is only 0 to 2 node or we can say that A tree doesnot contain not more than of 2 nodes is called Binary tree.



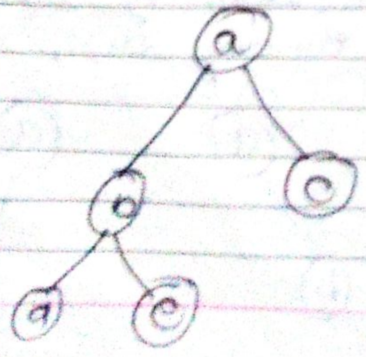


0,1,2

(Binary tree)

② Strictly Binary tree →

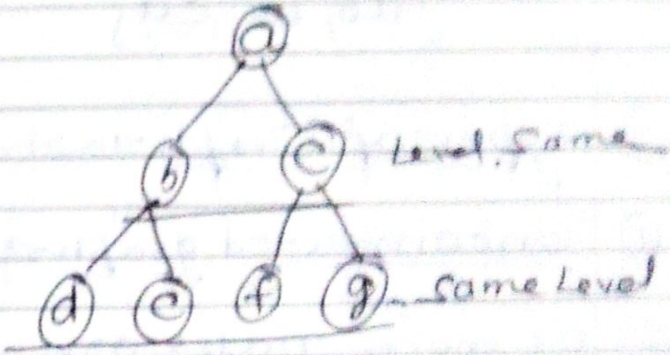
A binary tree is called strictly binary tree if ~~any~~ every node must have two or zero child. then this type of tree is called strictly Binary tree.



(0,2)

③ Complete Binary tree → (0,2)

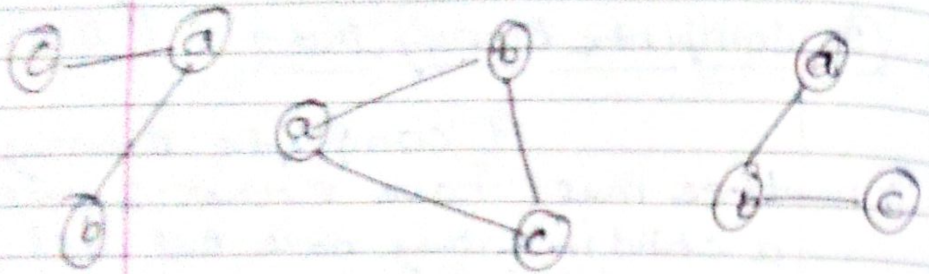
A complete Binary tree must have either zero or 2 child in every node But all the leaf nodes must have the same Level.



ⓑ Graph →

A graph (G = V,E) is said to be graph if it contain a set of vertices & pair of edges. called the Graph.

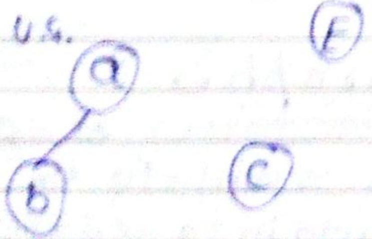
Here
(G = V, E)
where G = Graph
V = Set of vertices
E = Edges



V = {a, b, c}
E = {ab, bc, ca}

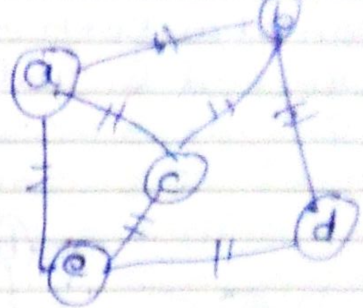
Types of Graph →

① Unconnected graph → When two or more nodes are not Connected is called U.G.



② Undirected graph :- When A Graph (G = V, E) has no proper direction of any Edge is called undirected graph. in Undirected graph, we can

move from down to up or up to down.



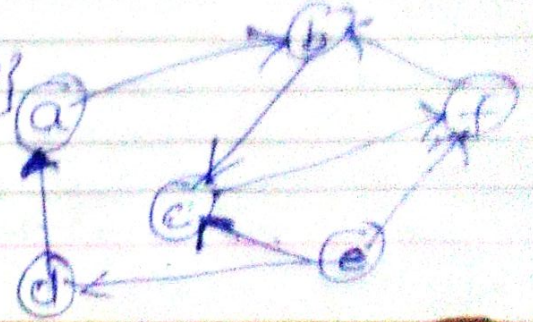
V vertex = {a, b, c, d, e}

E Edges = {ab, ac, ae, ed, bc, cb, ce, ec, de, ba, db, ea }

③ Directed graph or diagraph:- A graph (G = V, E) is called directed graph if each & every node ~~ha~~ must have a particular direc^n is called digraph

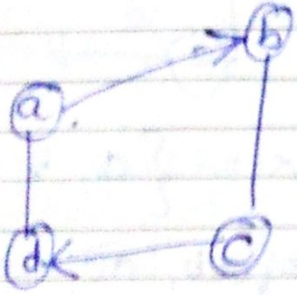
V = {a, b, c, d, e, f}

E = {ab, fb, bc, cf, ef, ed, da}



④ **Mixed graph :-** A graph(G = V,E) is called mixed graph, if it contain both the property of directed & undirected graph.

Ex.



V = { a, b, c, d }

E = { ab, cd, bc, cb, ad,da }

———x———x———x———

② IInd

Sparse Array :

Sparse array is a special type of an array. It is an array which contain zero and non-zero element but it contains a larger quantity of zero element. In other hands, we can say that it has approximately 80% of zero element.

Ex –

$$
\begin{array}{cc|ccccc}
 & & & & \text{Column} & & \\
 & & \text{I} & \text{II} & \text{III} & \text{IV} & \text{V} \\
 & \text{I} & 9 & 2 & 0 & 0 & 0 \\
 & \text{II} & 0 & 1 & 2 & 0 & 0 \\
\text{Row} & \text{III} & 0 & 0 & 0 & 0 & 0 \\
 & \text{IV} & 0 & 0 & 0 & 3 & 0 \\
 & \text{V} & 5 & 0 & 0 & 5 & 0
\end{array}
\quad \underset{5\times 5}{\text{Row column}}
$$

Representation of sparse array

Sparse Array can be represented in two ways :-

① Vector list Representation
② linked list "

① ~~Vector~~ list Representation →

when we represent our array with the row number & column number & value then it is the vector list Representation. It also contain the triplet value i.e <row no., column no., Non-zero value> of any sparse array.

Represent using vector list:-

Row No.	Column.No.	Non zero Value
1	1	9
1	2	2
2	2	1
4	4	3
5	1	5
5	4	5

② linked list Representation:-

linked list rep. is other important rep. of the sparse array. In this type of Representation, the linked list

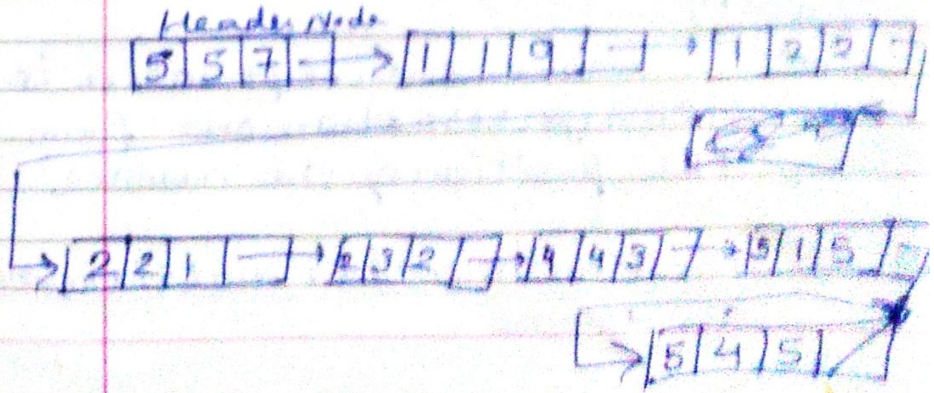
must contain the header node & in the header node it have

(i) No. of rows
(ii) No. of columns
(iii) No. of Non zero values

On the other hand, the remaining or forwarded nodes also have row number, column number & non zero value & also have a pointer which indicate the next node of the linked list. The remaining nodes have the following attributes:-

(i) Row number
(ii) Column numbers
(iii) Non-Zero value

Example -



One dimensional array :-

One dimensional array is the array in which an element in the array is represented only by one subscript is called one dimensial array. The subscript is the number which can be used ~~to~~ ~~at~~ for position or location of the number in the current position. The subscript value always written with in the brackets.

Example-

10	5	20	9	6
[0]	[1]	[2]	[3]	[4]

— One DIM

subscript indices

The subscript value is always less than one from the position of the number.

Two dimensional or multidimensional array-

	[1,1]	[1,2]	[1,3]
[1,1]	5	9	12
[2,1]	7	6	8
	[2,1]	[2,2]	[2,3]

Two dimensional array is the array when an element is rep. by subscript or indices in [row, column] form, this type of array is called two dimensial array. When an element has represented by more than two values in a single subscript is called multidimensial array.

Lower Bound in single dim. array

The lowest subscript in an one dimensial is called Lower Bound (LB).

5	4	9	7
[0]	[1]	[2]	[3]

The lowest subscript = [0]
So LB = 0

Upper Bound (UB) → The highest subscript in one dimensional array is called Upper bound.

$UB = [3] = \boxed{No.\ of\ element - 1}$

Range or length of the array

$R = UB - LB + 1$

The no. of elements present in one dimensional array is called Range or Length of the array z

$R = 3 - 0 + 1$
$R = \underline{\underline{4}}$

Address calculation for linear array →

5	8	2	6

Address calculation of linea array can be determine by the given formula :-

Address of element at

location A[P] = Base address + [P - LB] ×

where
A → array
P → location of the element
Base address → starting address of
LB → lower Bound
S → no. of bytes occupy the the element.

Example →

Consider an array, which contain 20 element. If address of 1st element is 1000 & each element is occupy 5 bytes in memory. Find out the position or location of A[6] element.

Soluⁿ- we have given,

Base address = 1000
S = 5
LB = 0
P = 6

From the formula -

Address of A[6] element

= 1000 * [6 - 0] × 5

= 1000 + [6×5]

= 1000 + 30

= 1030 Ans.

Example Consider in an array of 15 element in which address of Ist element is 500, & each element occupy 4 bytes in mem. Calculate address of 7 A(7) element in

Base address = 500
S = 4
LB = 0
P = 6

7	8	2	9

Address of A[6] element

= 500 + [6 - 0] × 4

= 500 + [6×4]

= 500 + 24

= 524 Ans

⇒ Difference b/w Data structure & Data type =

	Data type	Data structure
①	Data type is the type of data that can be hold by a variable in any programming lang.	It is a logical or mathematical model to assemble the data is called data structure
②	It helps us to	② It help us to arrange

	determine the type of data & other operations are performed on that data.	data & set of operⁿ performed on the data.
③	It is named group of data with similar charact e.g. integer, char, boolean and float	② It is work on the data type which help us to provide data ■ specified form.
④	A general form of class of data item is called data-types.	① An aggregate of data objects is called data-structure.

⇒ **opeⁿ on data-structure:**

The following operations can be done on data-structure are

① **Traversing :-**

It is the process which can be use to access & perform the data item individualy It can also be used to count the each data element exactly once

② **Sorting :-**

It is the process to arrange all the data in sequence when we perform sorting, mainly all the data can be sort in ascending form.

③ **Searching :-**

It is the process to search an element into the desired location in a list is called Searching.

If the desired element is found then searching is successful otherwise it is fail.

④ **Inserting :-**

It is a process to insert an element in to the proper position according to the user. By inserting an element, the total no. of elements is incremented by one.

⑤ **Deleting :-**

⑥ Merging:- It is a process to combined two same type of list into a single one.

Example:- list1 = | 9 | 8 | 19 |
Unsorted

list2 = | 3 | 5 | 6 |
Unsorted

merge list = | 9 | 8 | 19 | 3 | 5 | 6 |
Unsorted

⇒ Categories of data-structure:-

There are following categories of data-structure which are given below:-

① Primitive & Non-primitive data-structure:-

The data-structure which are atomic in nature or contain the property of indivisible elements called primitive data structure

These type of data structure are common in nature & operated by machine instruction. The data type such as integer, real, float, char are the primitive data structure. In any programming languages, the integer, Real, boolean, character are datatypes

✱ Non-primitive data structure:

The Non-primitive data-structure are not atomic (indivisble) in nature are called Non-primitive data structure. They are complex in nature. They are build by the help of primitive data structure. Some time they are deal with user-defined such as array, structure, stack, queue, & linked list are the example of Non-primitive data structure.

Example:-

int arrayname [size of array];
int number [10];
P NP

⇒ Homogenous data structure:-

The data-structure which ~~are~~ have same element contain the same data-type are called homogenous data-structure.

Example → Array

```
int marks[10];

char name [20];
```

* Hetrogenous data-structure:-

The data structure which have different element with different data types are called hetrogenous data-structure

Example:- Structure.

```
Struct Student
     {
 int roll no;
float marks;
Char name;
char address;
    };
```

⇒ static data structure:

10	20	18	15	–	–

— memory wastage of last 2 [illegible]

Static data structure is the data-structure when the program is used ~~at~~ compile time. The memory is not expand or not shrink so this type of data Structure has more memory wastage

Example:- Array data structure

* Dynamic data-structure:-

This type of data-structure is done on run-time. In this Structure, the memory is expand or shrink so that memory utilization is more & there or less wastage of memory.

Example:- pointer.

⇒ Big-oh Notation:

Big oh Notation

is a part of Asymptotic Notation that allow to measure of algorithms such as performance and/or memory requirement. It describe

① Algorithm performance as the number of element in a data structure increase

② Other behaviour such as memory consumption.

The algorithm used by various data structure for different operation like search delete & insert fall into constant time, linear o(n).

→ Complexity of an algorithm:

Different algorithm names are

Heap sort, merge sort, Quick, Radix sort, Bubble sort, insertion, Selection sort,

Complexity is " the quality or state of being complex " i.e so many varied interrelated components as a result in it difficult to understand.

So Complexity of an algorithm is the amount of time & amount of space require for an algorithm to solve any problem. complexity is of two type :-

① Space complexity :-

Space complexity is the amount of space taken by an algorithm to solve any problem. It has several possible solution with different space needed & to estimate the size of the largest problem that a program can solve.

② Time complexity :-

Time complexity is the amount of time taken by an algorithm to solve any problem. It describe several possible solution with different

Time requirement it also describe whether the program will provide a satisfactory real-time response.

*⇒ Analyisis and efficiency of algorithms

- Various algorithms are written to solve a specific problem and Algorithm that needs a minimum execution time and lest memory space is said to be to be an efficent algorithm

However inherent efficiency of an algorithm can't be modified irreslective of the cleverness of the programmer. for example to store an n×n array, n^2 memory location are neened and algorithm can Reduce this storge Requirement.

thus to select a suitable algorithm to solve a problem its execution shuld take lest time and and

minimum storge space while keeping the inherent efficieny of the algorithm

to ditermine the fallowing information is requred

* Time taken to Read the instrucation.

* Time taken to execute the instrucation.

* time taken to understand and interpret the instrucation

⇒ ⓐ Space-time trade off algorithm

the best algorithm to solve a given problam is one that required less space in memory and takes less time to complite its execution but in practis it is not always possible to achive both of this abjactive there may be more then.

one approach to solve a problem one approach may requir more space but less time to complete its execution. the second Approch may requir less space but take more time to complete execution one chuse first Approch if time is a constraint and second Approch if space is a constraint thus one may have to sacrifice at cost of the other. that is what we can say that there exists a time space tradeoff among algorithm

therefore, in most case the space and time required by an algorithm are inversely related are.

Space ∝ 1/time | $20x \propto \frac{1}{20x}$

e.g. ~~20x~~

one can reduce the space requirement by increasing running time or vice versa.

Example :-

Consider a simple example of computing the exponential function x^n where n is positive integer.

The function x^n is defined as:

$x * x * x * x$ - - - - n time

5^3 5×5×5

Solving this problem with two different

```
Step I   Read the variable x & n
     II  Assign the value x to y
     III Execute the step 4 for (n-1) times
     IV  Multiply the value of y & x & assign calculate value of y
     V   write the value of y.
     VI  stop
```

⇒ String :-

String is a sequence or collection of characters that is treated as a single data item

→ * A String is a structure for Storing text it is a finite Sequence of zero or more characters or symbols. Strings are mostly used for displaying text to the screen.

for Example :→

* 'computer' is a string with length

* '' is a string with length 0

* 'Suman wadhwa' is a string with length 12. her blank Space is also treated as a character and hence adds to the length of the string.

Q Useful of String' :→

Strings are useful in the following ways,

1. Text handling :→ One of the most common uses of strings is text handling. Text handling facilities include newspaper editing and typesetting, book and report generation, computer programs and data preparation.

2. Lexical Analysis :→ Strings are also used for lexical analysis. A module in a compiler separates the source input into basic word like constructs such as identifier names, numes, numeric constants, string constants, keywords, operators etc, is commonly called the scanner or lexical analysis.

3. Word Frequencies :→ Strings can be used to determine the frequencies of words in a piece of text.

4. **String searching** :→ A common problem is to find the first or all occurrences of a strings termed the pattern P, in a text string T. Algorithms for solving this problem have applications in virus checkers, editors document processors and information retrieval systems.

Operations on Strings :→ The common operations that can be handled/performed on a string include the following :-

1. Create a new string of text
2. Concatenate two strings to form a third string.
3. Search and replace a substring within a string
4. Identify a string.
5. Compute the length of a string
6. Insert a word withing a string
7. Delete a word from a string

1. **Create a new string of text** :→ Creation of a string means to construct a representation of a string. It enables to return the value of a string in a variable

2. **Concatenate two strings to form a third string** :→ Concatedion means positioniting of string of string elements. It is a binary operation. This operation is used to combine to two or more character string to create new strings.

3. **Search and replace a substring within a string** :→ This operation is used to obtain a substring from a given string. For obtaining a subtstring from a given string

the following information is required.

4. Identify a String :->

To identify a string these steps are followed:

(i) The given substring is searched in the strings on a character by character basis from the first character of the string to the last.

5. Compute the length of a String :->

The number of characters in a string is called its length. It has one operand of type string and give a result of type integer.

6. Insert a word within a String :->

Insertion means to insert a given string in the middle or in a required position of another string.

(a) Let p be the position where insertion operation should take place.

7. Delete a word from a string

Deleting a word from a string means to remove a sequence of characters starting from a particular position

String Matching :- While editing a text, finding the occurrence of a pattern is called string matching. In other wards, string matching is detecting the occurrence of a particular substring or a pattern, in another string, called the text.

The text is a document being edited and the pattern searched for is a particular word which is supplied by the user. To aid the text editing programs, many efficient algorithms have been developed.

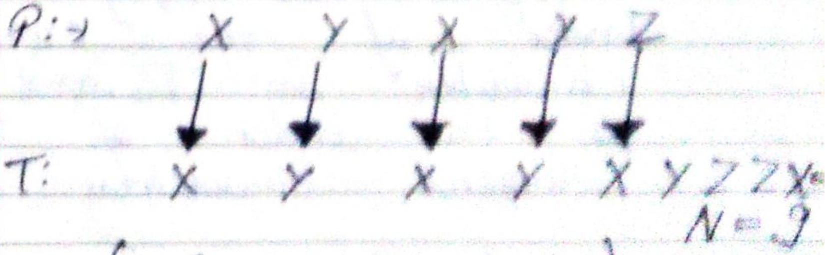
A Straightforward String Matching :-

Let P be the pattern to be searched in the text T. Let m be the length of pattern P and n the length of text T where n is fairly larger than m.

characters in P and T are denoted by subscripted P's and t's respectively.

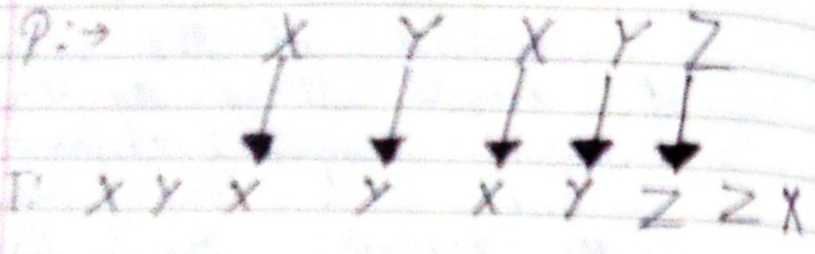
Starting at the beginning of each string, the algorithm compares characters, one after the other until either the pattern is executed or a mismatch is found.

Comparisons are done in left - to right order. The following comparison explains the straigthforward string matching :-



(First comparison.)

The second comparison finds mismatch at position X-Y.



(Third Comparison)

The third comparison is successful match.

Algorithm :→

Let i is the current guess at where P begins in T, j is the index of the current character in T and k is the index of the current character in P.

```
Step 1: Set i ← 1; j ← 1; k ← 1.
Step 2: While j ≤ n and k ≤ m
        repeat step 3.
```

```
Step 3: If $t_j$ = $p_k$
          then
            (i) Set j ← j + 1
            (ii) Set k ← k + 1
        else
            (i) Set i ← i + 1
            (ii) Set j ← 1
            (iii) Set k ← 1
Step 4: If k > m
          them
            Match found at
        position starting i.
          else
            Match not found
Step 5: Stop.
```

⇒ Boyer - Moore algorithm:-

The most efficient String matching algo is Boyer-moore algorithm which is used application. It was developed by S. Boyer & J. Strother moore

Some characters may be skipped over entirely. For longer pattern, this is faster & good algorithm bcz it jump to next place in the text leaving the place where the pattern can't appear.

- It scan the characters of pattern from rt. to left starting from the rightmost characters. when there is a mismatch, it use two m/d to shift the window to the right.

Example →

text S: a b a a b c a b b c b a b b a b a b b

pattern b a b a b b

step-I – Make the numbering of the given text & do matching from rightmost of the string.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

S: a b a a b c a b b c b a b b a b a b b a

p: b a b a b b

from the matching we get,

s[5] ≠ p[5].

The ~~string~~ pattern character cannot contain the c character so we skip the position from 0 to 5.

step-II

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

S: a b a a b c a b b c b a b b a b a b b a

p: b a b a b b ← skip

p: b a b a b b

S[11] = a & b[5] = b so

S[11] ≠ b[5], so we jump to 2 position of rt to the string.

step-III

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

S: a b a a b c a b b c b a b b a b a b b a

p: b a b a b b

S[9] = c & p[1] = a so S[9] ≠ p[1]

Step

```
     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
S:   a b a a b c a b b c b  a  b  b  a  b  a  b  b  c
                                     ↓  ↓
p:                          b  a  b  a  b  b
```

S[15] = p[5] but S[14] ≠ p[4]

⇒ Here is the second comparison a mismatch is produced. Here S [14] = a and P [4] = b.

Therefore S [14] ≠ P [4]. The pattern is shifted so that the rightmost 'a' i.e P [3] in the pattern is aligned to text symbol a i.e S[14]. Therefore the pattern is shifted to position 11.

(v)

```
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
S  a b a a b c a b b c b  a  b  b  a  b  a  b  b  a
                          b  a  b  a  b  b
                                         ↓
```

(P)

In the first comparison a mismatch is produced. Here S [16] = a and P [5] = b

Therefore S [16] ≠ P [5]. The pattern can be shifted so that the rightmost 'a' i.e. P [3] in the pattern 'P' is aligned to text symbol a ie S [16] Thus the pattern is shifted to position 13.

(vi)

```
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
S  a b a a b c a b b c b  a  b  b  a  b  a  b  b  a
                                ↓  ↓  ↓  ↓  ↓  ↓
                                b  a  b  a  b  b
```

(4) Here all the comparisons ends with a success.

Thus in Boyes-Moore Algorithm:

- The pattern is searched from right to left
- The pattern is shifted to the right whenever a mismatch is encountered. The amount of shift depends on the character of the text at the text at the time of a mismatch

O(n/m) is the best performance

Representation of two dimensional array in memory:-

We can calculate using row major order & column major order

Row Major order = location of A[i][j]

Base address + $[n.(i-lBR)+(j-lBC)]$ ×w

where Base address = Address of first element
i = row no.
lBR = lower bound array row
j = column no.
lBC = lower bound column
W = no. of byte occupy
n = no. of column

Column major order:-

Address of A[i][j] element:-

Base address + $[(i-lBR)+m(j-lBC)]$×w

Example:-

Suppose an array contain 10 rows 10 columns. The first element has 1000 location, & it occupy 4 bytes in memory find the address of A[6][7] element

Soln - Given B.A = 1000
w = 4
p = 6
q = 7
m = 10
n = 10

Row major order =

Address of A[6][7]th elements

$= 1000 + [10(6-0) + (7-0)] \times 4$

$= 1000 + [10(6) + 7] \times 4$

$= 1000 + [60 + 7] \times 4$

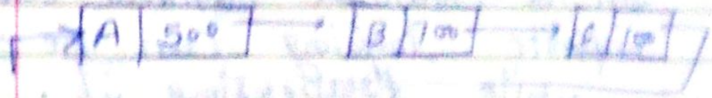
$= 1000 + (67 \times 4) = 1000 + 268$

$= 1268$

⇒ Threaded list :- A list data structure is called threaded list in which last

element of each list contain the location of first element of the list & which specify the terminal member, called threaded list.

Example -



Threaded list

Advantage of linked list :

① It is use for repeatedly go around the list.

② Accessing of any node is much faster.

③ Any node can be start node.

④ implementation of queue.

⑤ One could start at any node to traverse the entire list.

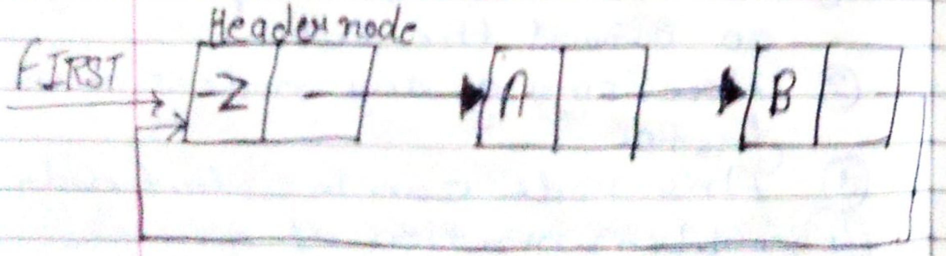
⑥ Since the pointer of last node points to the head node, one can move from last node to the head node very efficently.

*

Q (c) Describe the importance of Head node in a circular linked list.

MDU BCA July 2009, Jan 2008, July 2006

In a circular linked list, an infinite looping can be caused while traversing it.
The way to avoid this infinite loop condition is to place a header node at the beginning of circular linked list as shown below :-



- A record not consistent with the other records of the list.
- Some information about the list.

For example, if the data recorded on different nodes of the linked list are positive integers their the header node may have a negative integer recorded on it. This difference in the type of data recorded helps in identifying the beginning of the list.

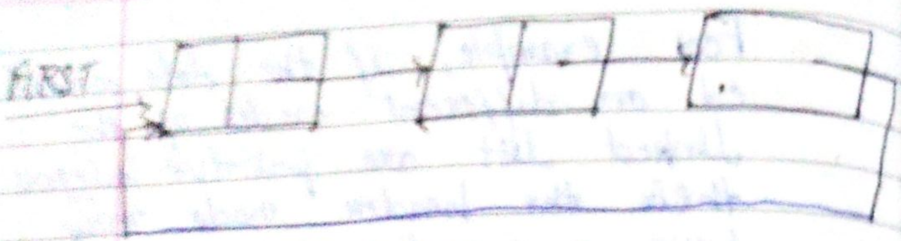
Q 10 (a) Describe circular linked lists and their application with examples

MDU BCA Dec 2014

Circular linked list :-

Circular linked list is linked list in which the last node points to the node at the beginning of the list

Thus the link of the node is not a null pointer but it contains the address of the first node.



Applications of Circular Linked List:-
Following are the applications of circular linked list :-

1. Circular linked lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them giving each of them a slice of time to execute, and then making them wait while the ~~CU~~ CPU is given to anthoer application.

2 It saves time when when one has to go to the first node

from the last node It can be done in single step because there is no need to traverse the other nodes.

3 Useful for implementation of queue.

4. Since the pointer of last node points to the head node, one can move from last node to the head node very efficiently

5. Each node in a circular linked list can be accessed from any node whereas a singly linked list can be accessed only by starting from the first node

Ques-10(b) Explain Doubly Linked list, write an algorithm for cealion of a doubly linked list of n nodes.
MDU BCA Jan 201

Doubly Linked List :-
A linked list in which each node has not only a pointer to the next

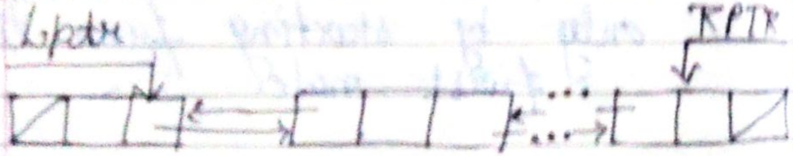
node but also a pointer to the prior node is called doubly linked list.

Node

LPTR	INFO	RPTR

Here LPTR → Left link field.
RPTR → Right link field
INFO → Information field.

Doubly linked list is shown below



LPTR and RPTR are pointer variables that denote the left most and rightmost nodes in the list respectively.

The left link most node is null and the right link of the right most node is null.

In C language, the node of a doubly linked list is defined as:-

```
Struct dnode
 {
  Struct dnode *lptr;
  data_type info;
 Struct dnode *rptr;

 };
```

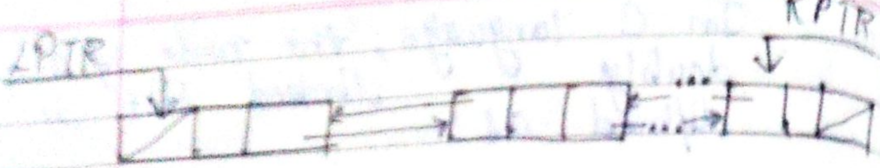
* Doubly Linked List :- A linked list in which each node has not only a pointer to the next node but also a pointer to the prior node is called doubly linked list.

Node

LPTR	INFO	RPTR

Here LPTR → Left link field
RPTR → Right link field
INFO → Information field

Doubly linked list is shown below :-



LPTR and RPTR are pointer variables that denote the leftmost and rightmost nodes in the list respectively.

* Inserting a node in a Doubly Linked List :-

Inserting a node in a doubly linked list is done by acquiring a node from the availility list and adjusting the pointers appropriately.

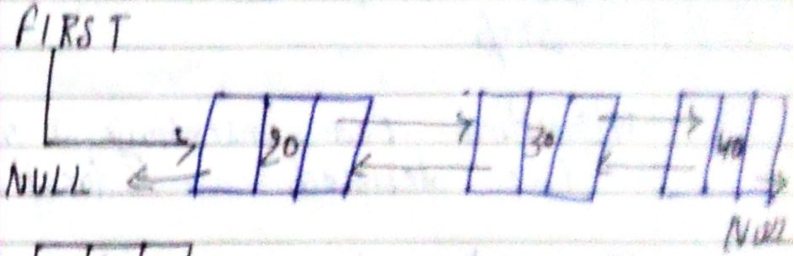
(i) At the beginning
(ii) At the desired location.
(iii) At the end.

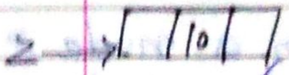
1) Algorithm for Inserting an Acquired Node at the beginning :-

The following algorithm acquires a node Z and inserts

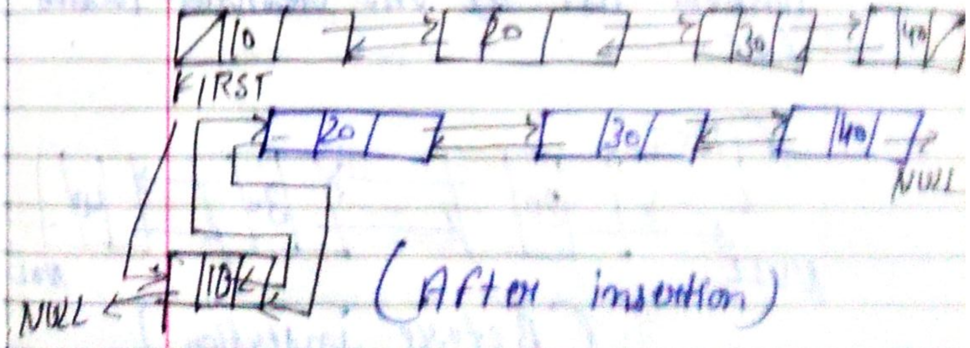
it at the beginning of a doubly linked list

A pointer FIRST at the beginning is pointing the list





(Before insertion)



(After insertion)

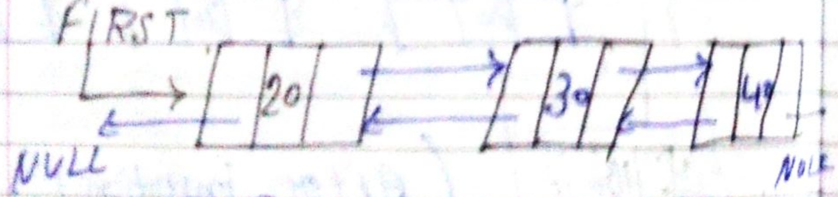
Steps of algorithm are :-

Step1 Set Z ← AV AIL
Step-2 Set INFO (Z) ← DATA
Step-3 Set LPTR (Z) ← NULL.

```
Step 4  Set RPTR (Z) ⇐ FIRST
Step 5  IF (FIRST is not equal to NULL)
              then
                    Set LPTR (FIRST) ⇐ Z

Step 6  Set FIRST - Z
Step 7  Stop
```

(2) Algorithm for inserting a node at the desired location :-

In this algorithm a node Z is inserted in the doubly linked list at the desired location LOC.



(Before insertion)

FIRST

⇒ Application of linked list :-

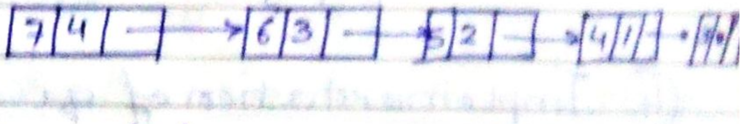
① It is implement other data structure like, stack, queue, tree & graph.

② To perform polynomial manipulation

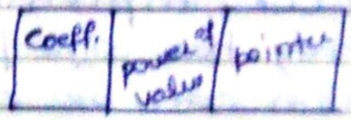
A polynomial of the type

$7x^4 + 6x^3 + 5x^2 + 4x + 3$

may be represented in the form of linked list



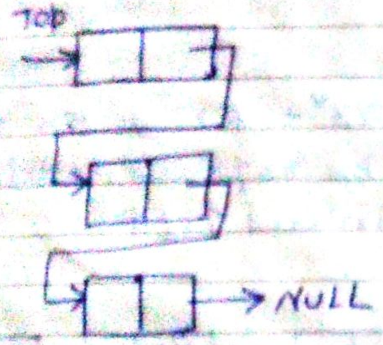
General form -



coefficient = value.
power = exponential
pointer = link to the next node.

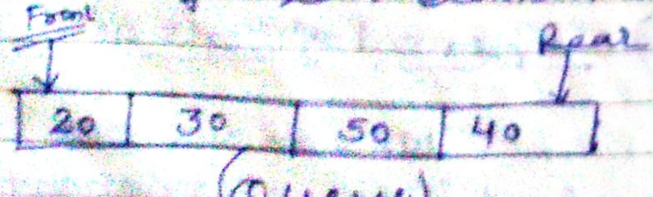
③ implement of stack as a linked list

We can use our linked list in the implementation of stack & The stack is full, No element can be inserted so with the help of linked list, we overcome this problem.



④ Implementation of queue as a linked list →

Queue implementation using array has size problem. This problem can be overcome using linked list. with the help of linked list, we use front for first element added & Rear for last element added.



(Queue)

Que-11(b) Describe garbage collection and its advantages with examples.
MDU BCA Dec 24

* Garbage Collection:-

During the program execution some blocks of storage are needed but the same at some later stage become unnecessary and remain unused such blocks are called garbage.

* How to recover these garbage blocks? One method for collecting the garbage blocks is to leave no storage block free until there is almost nothing left. Then all the allocated blocks are checked and those that are no longer in use are freed.

* Problems is the context of storage release :-

The problem is that of dangling references. A dangling reference is a pointer existing in a program which still accesses a block that has been freed.

* Garbage collection algorithm:

The garbage collection algorithm generally has two phases.

(i) Marking Phase:- In the first phase known as marking phase, all blocks in use are marked.

(ii) Collection Phase: In the second phase known as collection phase the collection algorithm scans through all the blocks and all unmarked blocks which are not being used are returned to free list.

Advantage of Garbage collection

Following are the advantages of garbage collection:

1) It ensures program integrity for example, garbage collection is an important part of Java's security strategy.

2) It increases productivity. One need not worry about memory issues.

3.) The manual memory management done by the programmer is time consuming and error prone. Hence the automatic memory management is better.

4.) Reusability of the memory can be achieved with the help of garbage collection.

Q15(b) what is circular queue? Discuss its advantages over linear queue. MDU BCA May 20

* Circular Queue :→ A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full.

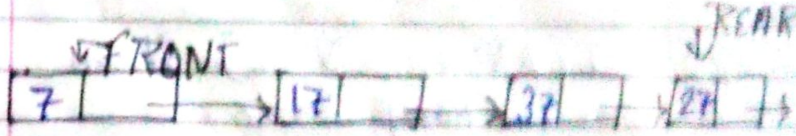
*. Advantages of Circular Queue over Linear Queue :- In simple queue when the elements are deleted from the front, the free space cannot be utilized again but in the circular queue when the rear at the maximum position it again starts filling the element from the starting of the queue.

15C) Explain the linked representation of queue with examples.
MDU BCA Dec. 2013

OR

How do you overcome the limitations of queue using array? Also write algorithm to insert element into linked queue and delete element form linked queue.

*. Linked representation of queue :- Queue implementation using array has size problem. The problem is overcome by implementing of queue using linked list.



The node of linked queue is defined as struct qnode

```
{
    int info;
  Struct qnode *next;
};
```

* Algorithm of insert an element in a linked queue:-

Let VAL be the element to be inserted in a queue having FRONT and REAR as the pointers containing the addresses of the front and rear elements.

```
Step 1: Z ← AVAIL
Step 2: Avail ← NEXT(AVAIL)
Step 3: INFO (Z) ← VAL
Step 4: NEXT(Z) ← NULL

Step 5: If (FRONT = NULL) them
            FRONT ← Z

        else
          (i) NEXT (REAR) ← Z
          ii) REAR ← Z

Step 6:- Stop
```

* C- Implementation of Insert operation :-

```
Insert (struct qnode ** front, struct
qnode ** rear, int val)
 {
   struct qnode * z;

z = (struct qnode *) malloc (size of
(struct qnode));

z -> info = val;

z -> next = NULL;

if ((* front) == NULL;

  (* front) = z;

else

(* rear) -> next = z;

(* rear) = z;

    }.
```

* Algorithm to delete an element from a linked queue:-

Let FRONT and REAR are the pointers containing the addresses of the front and rear elements. AVAIL is a pointer to the first node of the availability list. Z is a temporary pointer. Steps of algorithm are:-

⇒ operations on stack :-

Various operations on stack are

① Create (S) :-

initially, we create the stack. From creating the number of elements in stack is zero.

Ex

Nod(Create (S)) = zero.

② Isempty(S) :-

This operations determine whether the stack is empty or not. this operⁿ give only two value either true or false. If no element in the stack, it give true result otherwise false.

Ex

isempty(Create(S)) = True

③ Isfull(Stack) :- This operⁿ give whether our stack is full or not. like as, Isempty operation it give also two value either true or false, when it give false value, we can insert one or more elements in the stack otherwise this give true Result.

Ex

Isfull (Create(S)) = False

④ Insert(push(element, stack)):-

This oper^n insert an element into the stack, after every push operation, the no. of element is incremented by one.

example :-

push(stack, d) = d

⑤ Pop :-

This operation remove an element from the stack. From different to the push operation, every pop operation decrement the no. of elements by one.

⑥ Peek :- This operation describe the top element from the stack, After every push & pop operation, the peek element must be change

⇒ Application of stack :

① Matching parenthesis :-

It is use to match the parenthesis in a programming language. The compiler check the whole program from Left to right. If there is a Left parenthesis, then in further we get the right parenthesis then there is no error condition bcz we pop the already available Left parenthesis. If not Left parenthesis is found then there is an error encountered.

② Recursion :-

Recursion is an important application of stack in which a procedure or a function called itself in a program then it is recursive procedure & the process is called recursion.

Example-

$n! = n(n-1)(n-2)(n-3)(n-4)!$

③ Converting infix to postfix notation

infix = A ± B ⇒ A,B = operand.

Postfix = A B +   +, -, *, %, ↑ symbols

Prefix = + A B   operator

⇒ Rules for Converting infix notation into postfix notation:
Scan the exp. from left to right.

① If a left parenthesis is found then push into the stack

② If any right parenthesis is found, then all the symbol occur before the left parenthesis is pop.

③

④ The operand is directly goes to the postfix notation,

L.P | H.P = No change
H.P | L.P = change

Higher priority	↑	B
Medium "	/, *	O
Lower "	+, -	D
		M
		A
		S

Ex 2+5-4%.2*10/5
= 2+5-2*10/5
= 2+5-20/5
= 2+5-4
= 2+1
= 3 (correct ans.)

Suppose our expression is

A+B*C. Convert it into postfix
1 2 3 4 5

Symbol scanned	Stack	Postfix Notation
A		A
+	+	A
B	+	AB
*	+*	AB
C		ABC
		ABC*+

Given Expression

P*(Q+(R+S)*(T+U)/V)*W

	Symbol Scanned	Stack	Postfix Notation
1	P		P
2.	*	*	P
3.	(	*(	P
4	Q	*(	PQ
5.	+	*(+	PQ
6.	(	*(+(	PQ
7.	R	*(+(	PQR
8.	+	*(+(+	PQR
9.	S	*(+(+	PQRS
10.	)	*(+	PQRS+
11	*	*(+*	PQRS+
12.	(	*(+*(	PQRS+
13.	T	*(+*(	PQRS+T
14.	+	*(+*(+	PQRS+T
15.	U	*(+*(+	PQRS+TU
16.	)	*(+*	PQRS+TU+
17.	/	*(+/	PQRS+TU+*
18.	V	*(+/	PQRS+TU+*V
19.	)	*	PQRS+TU+*V/+
20.	*	*	PQRS+TU+*V/+*
21.	W	*	PQRS+TU+* V/+*W
			PQRS+TU+*V/+*W

Ex (A*B+C/(D*E ↑F))/G

Symbol Scanned	Stack	Postfix Notation
(	(	
A		A
*	(*	A
B	(*	AB
+	(+	AB*
C	(+	AB*C
/	(+/	AB*C
D	(+/	AB*CD
*	(+*	AB*CD/
E	(+*	AB*CD/E
↑	(+*↑	AB*CD/E
F	(+*↑	AB*CD/EF
)		AB*CD/EF↑*+
/	/	AB*CD/EF↑*+
G	/	AB*CD/EF↑*+G
		AB*CD/EF↑*+G/

A*B-[C+D)-(E-F)+H/F↑I

Symbol Scaned	Stack	Post fix notation
A		A
*	*	A
B	*	AB
-	*- -	AB*
[	*-[	AB
C	*-[	ABC
+	*-[+	ABC
D	*-[+	ABCD
)	*-	ABCD+
-	*—	ABCD+
(	*-(	ABCD+
E	*-(	ABCD+E
-	*-(	ABCD+E-
F	*-	ABCD*E-F
)	*	ABCD+E-F
+	*+	ABCD+E-F
H	*+	ABCD+E-FH
/	*+/	ABCD+E-FH/
F	*+	ABCD+EFH/F
↑	*+↑	ABCD+EFH/F
I	*+↑	ABCD+EFH/FI

Que 15(a) what is circular queue? what operations are applied on it? Explain its linked representation with examples

[illegible]

* Circular Queue :→
A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full.

* Operations on Circular Queues :→
operations performed on circular queue are the basic operations such as insertion, deletion of the elements.

1) To insert an element in a circular queue :- The insert operation for array implement circular queues involves the following tasks :-

(•) Checking whether the circular

queue is already full

(•) updating the rear pointer

(•) Inserting the new element at the rear location.

* Steps of Algorithm :-

```
Step 1: If (Front = 0 and Rear = n-1) or
        (Front = Rear +1)

    (a) write "CIRCULAR QUEUE FULL".
    (b) goto Step 4.

Step 2: If (Front = Rear = 1) then

    (a) Set front ← 0
    (b) Set Rear ← 0
    (c) Set CQ[Rear] ← val

Step 3: If (Rear = n-1) then
    (a) Set Rear ← 0
    (b) Set CQ [Rear] ← val

    else
    (a) Set Rear ← Rear +1
    (b) Set CQ [Rear] ← val
```

Step 4: Stop.

2. To remove an element from a circular queue:

In this algorithm, a circular queue CQ is being used which can store maximum n elements. The circular queue elements are CQ [0], CQ [1], ..., CQ [n-1]. This algorithm deletes an element from the circular queue CQ and assigns it to a new variable 'val'.)

* Steps of Algorithm :-

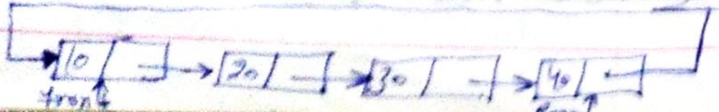
```
Step 1: If (Front = Rear = 1) then
        (a) write "CIRCULAR QUEUE EMPTY
        (b) goto Step 4.

Step 2: If (Front = Rear) then
    (a) Set val ← CQ (Front)
    (b) Set front ← 1
    (c) Set Rear ← 1.
```

```
Step 3  If (front=n-1) then
  (a) Set Val ← CQ [front]
  (b) set ~~back~~ ← 0

  else
  (a) set Val ← CQ [front]
      set front ← front+1

Step-4: Stop.
```

* Linked representation of circular queue.

Consider the queue of integers 10, 20, 30, 40. in the linked implementation the memory cell containing the first element will contain the pointer to the second element the memory cell containing the second element will contain the pointer to the third element etc.

The node and arc diagram representing circular queue of above example is show below



⇒ Convert the following postfix into infix notation.

Given expression

A = 40, 5, 2, ^, 12, 3, /, +, −

Symbol scanned	Stack
40	40
5	40, 5
2	40, 5, 2
^	40, 25
12	40, 25, 12
3	40, 25, 12, 3
/	40, 25, 4
+	40, 29
−	11

Answer = 11

⇒ Converting infix notation into prefix notation.

Rules →

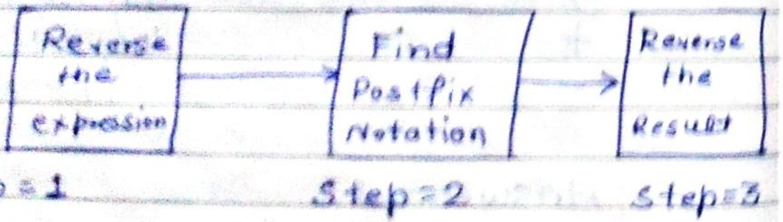
1. write the Given expression

into Reverse order

2. After find the reverse exp., Take the postfix notation of the find expression.

3. After converting into postfix notation, result must be write again into reverse order inorder to obtain the prefix notations.

Using diagram:-



(P+Q*C-D)/(E*F)

Step-I (F*E)/(D-C*Q+P)

Step-II

Symbol Scanned	Stack	Postfix
(	(	
F	(	F
*	(*	F
E	(*	FE
)		FE*
/	/	FE*
(	/(	FE*
D	/(	FE*D
-	/(-	FE*D
C	/(-	FE*DC
*	/(-*	FE*DC
Q	/(-*	FE*DCQ
+	/(+	FE*DCQ*-
)	/	FE*DCQ*-P+
		FE*DCQ*-P+/

Step 3 ~~P+-*QCD*EF~~

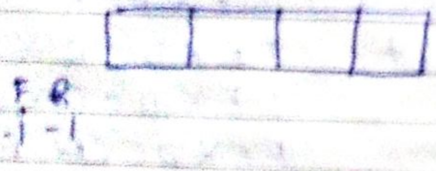
Result is

/+P-*QCD*EF

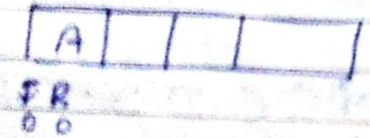
Disadvantages of linear Queue

The only disadvantage of linear queue is that if the last position is occupied by the some element in linear queue, we cann't insert a new element in the front of queue even if the front of queue is empty.

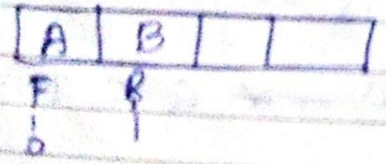
Example -

①


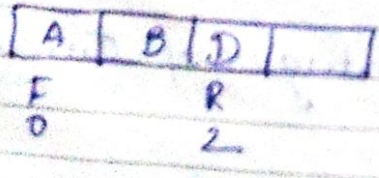
← insert A

②


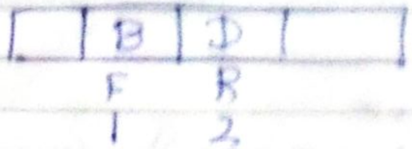
← insert B

③


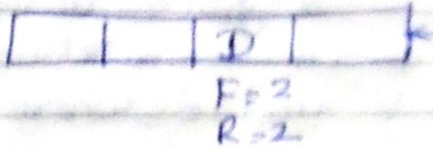
← insert D

④


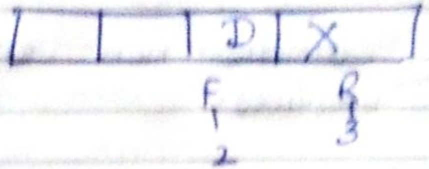
← Delete A

⑤


← Delete B

⑥


← insert X

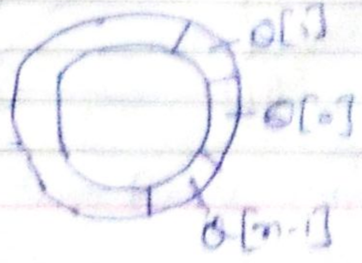
⑦


← insert Y

Queue overflow

The disadvantage of linear queue can be resolve by the help of circular queue.

A circular queue is a queue in which we can easily insert a new element in the first or second position even if last position in circular queue is occupied by some element.



Q. (a)

Ans- Graph :⇒

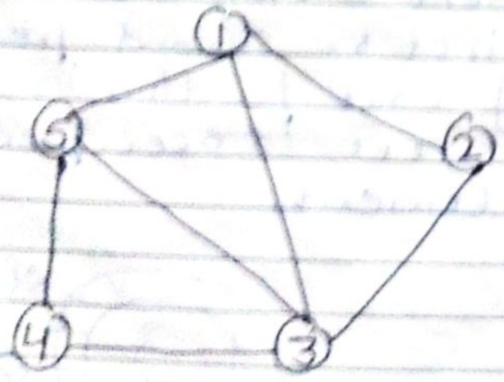
A graph G is defined that which Consists of two Sets V and E where

- V is finite non empty set of vertices.

- E is set of pair of vertices these pairs are called edges.

V(G) represents the set of vertices of graph G. E(G) represent the graph.

e.g. Consider the following graph G



Here: V(G) = {1, 2, 3, 4, 5}

$E(G) = \{(1,2), (2,3), (3,4), (4,5), (5,1), (1,3), (3,5)\}$

Applications of Graphs :

Graphs are widely used in Computer science. There are many applications of graph some of them are.

1. Graph are theory is widely used in the engineering applications such as network analysis, artifical intelligence, Computer writing, Computer graphics etc.

2. Graphs are used in Computer networking such as Local Area Network (LAN), Wide Area Network (WAN), internet working.

- Such shortest path problem.
- Topological sorting of graph.
- Connectivity in a graph.
- Spanning ~~m~~ trees.

- Euler's path, Hamiltionian path etc

Q: (b)

ans- Explain the following terms of graph:

Graph types, Degree of a vertex, path and length of path, self loop, parallel edges, isolated node.

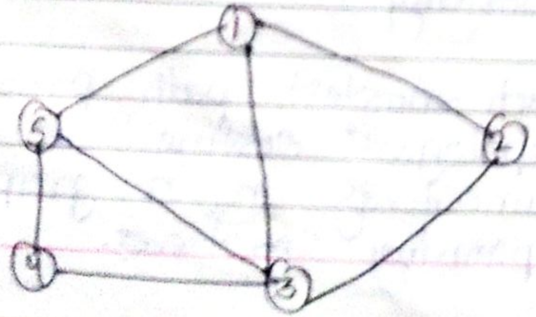
- Graph Types!

Following are the types of graph:-

1. Undirected Graph:-

An undirected graph is that in which there is no specific direction associated with the edges.

eg The given below is an undirected graph!



2. Directed Graph OR Digraph:-

A directed graph is that in which each edge has specific direction.

In a directed pair (v₁, v₂)

- V₁ is called the tail or
- initial vertex.
- V₂ is called the head or final vertex.

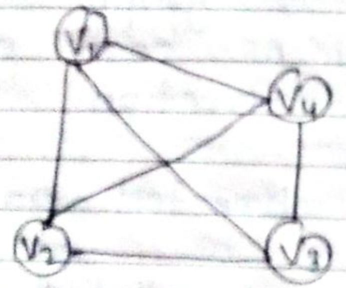
3. Mixed Graph:-

A Graph is called mixed graph if some of the edges are directed and some are undirected.

- Degree of a vertex is

The degree of a vertex (u) is the number of edges incident to that vertex (u). It is denoted by d(u).

e.g Consider the following graph(s)



$d(V_1) = 2$
$d(V_2) = 2$
$d(V_3) = 3$
$d(V_4) = 3$

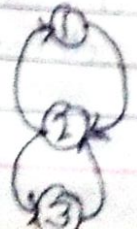
⇒ In - degree and Out - degree:

When G is a directed graph:-

- The in - degree of a vertex V is defined to be the number of edge for which V is the head.

- The out - degree is defined to be the number of edges for which V is the tail.

e.g Consider the following directed graph:



Vertex 1 has:- In degree 1 and out degree 1.

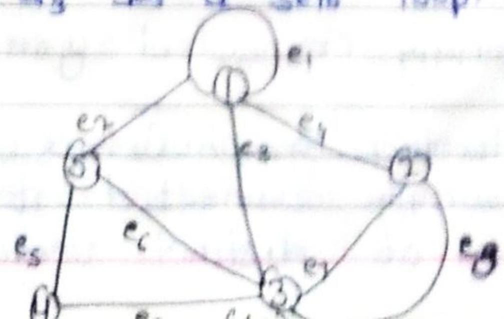
Vertex 2 has:- In - degree 2 and Out - degree 2.

Vertex 3 has:- In - degree 1 and Out - degree 1.

- Path and length of path:- A path from vertex a to vertex b of length n is defined as a Sequence of (n+1) vertices:
$a = V_1, V_2, V_3 \ldots \ldots V_n, V_{n+1} = b$

- Self - loop :⇒ An edge (V_i, V_i) is Called Self loop.

e.g Edge e_1 in the given graph G_3 is a self loop.



- Parallel - Edges :-

A pair of edges which has the same end vertica are called parallel edges.

eg Edges e_7 and e_8 in graph G_1 are parallel edges.

Q:(c) Explain the following :

Complete graph or fully connected graph, strongly connected graph, Simple graph, multigraph, null graph.

- Complete graph OR fully connected graph :⇒

An n vertex undirected graph with exactly $n(n-1)/2$ edges is said to be complete graph.

- Strongly Connected graph :⇒

A directed graph G is said to be strongly connected if for every pair of distinct vertices V_i, V_j in V(G) there is a directed path from V_i to V_j and also from V_j to V_i

- Multigraph :-

A graph which have parallel edges is called a multigraph.

- Simple - Graph :

A graph having neither self - loop nor parallel edges is called simple graph

- Null. Graph :-

A graph is called null graph if it contains only isolated nodes.

Q:(d)

ans - Graph - Traversal :-

A graph traversal means visiting all the nodes of the graph.

Graph traversal is needed in Several applications like :

- To Count the number of nodes
- To find total distance between cities.

1. Depth first Search :⇒

The procedure for depth first search of an undirected graph is as follow :-

(a) first, the start vertex V is visited and marked.

(b) Next an unmarked vertex W adjacent to V is selected and marked. This W vertex becomes the new start vertex.

2. Breadth first Search :⇒

The procedure for breadth first search of an undirected graph is as follow :-

Starting at vertex V and marking it as visited, all univisited vertices adjacent to V are visited and marked. Next repeat the process from an adjecent vertex W which is one of the visited and marked vertices.

Q: 2-(c)

ans - Differentiate between DFS and BFS

DFS	BFS
1. Depth first Search (DFS) explores all of the nodes reachable from X before moving on to any of X's siblings i.e this search uses a strategy that searches "deeper" in the graph.	1. Breadth first Search (BFS) explores siblings before children i.e this search discove all vertices at distance K from the source before discovering any vertices at distance K+1.
2. Backtracking is possible from a dead end.	2. Backtracking is not possible.

	DFS	BFS
3	Search is done in one particular direction	3. The vertices in the same level are searched parallely.
4	DFS places discovered vertices in LIFO stack, exploring vertices as discovered	4. BFS places discovered vertices in FIFO queue exploring vertices in the order discovered ,

Qus: 4(a)

ans- Representation of graphs in memory

Two represtations are commonly used . These are :

1. Sequential representation by means of adjecency matrix.

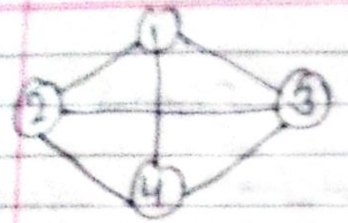
2. Linked representation.

1. Adjacency matrix representation :

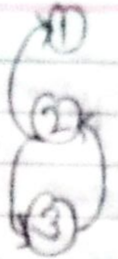
Let a graph G(V,E) with n vertics . The adjecency matrix A, of G is a two dimensional n × n array with the property :

$$A[i, j] = 1 \text{ if the edge } (V_i, V_j) \text{ is in } E(G)$$
$$= 0 \text{ if the edge } (V_i, V_j) \text{ is not in } E(G).$$

e.g Consider the following two graphs G_1 and G_2



(G₁)



(G₂)

Adjacency matrices of graph G_1 and G_2 are:-

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 1 & 0 \end{array}$$

(For G_1)

$$\begin{array}{c|ccc} & 1, & 2, & 3 \\ \hline 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 \\ 3 & 0 & 1 & 0 \end{array}$$

(For G_2)

The following points are clear from these adjacency matrices:

- The adjacency matrix for an undirected graph is symmetric, as the lower and upper triangle are same.

e.g. As in graph G_1.

- The adjacency matrix for a directed graph need not by symmetric.

⇒ Disadvantages :⇒

The disadvantages of adjacency matrix are:

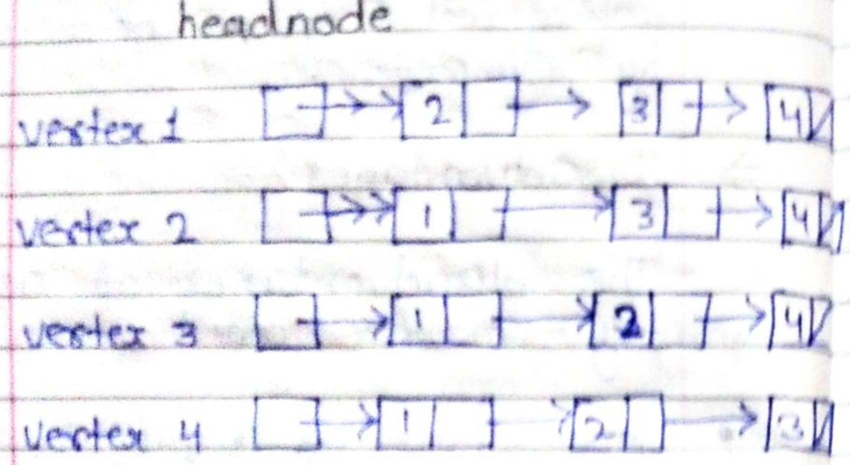
- More space is required to represent a graph. It takes n^2 space to represent a graph of n vertices.

The time required is n^2 for most of the graph problems.

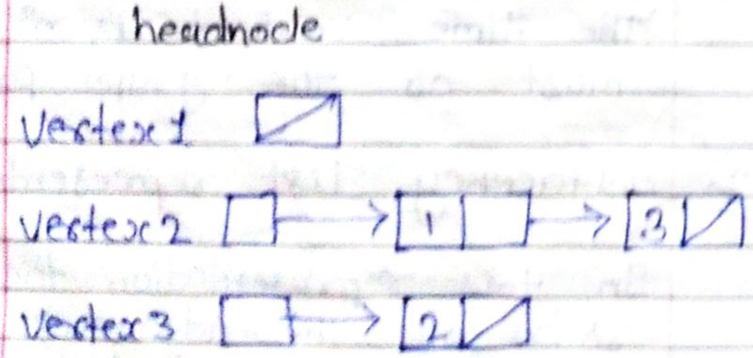
2. Adjacency list representation:

In this representation the n rows of the adjacency matrix are represented by n linked lists. There is one linked list for each vertex.

- For graph G_1,



- For graph G_2



There is a headnode for each list which provides sequential random access to the adjacency list.

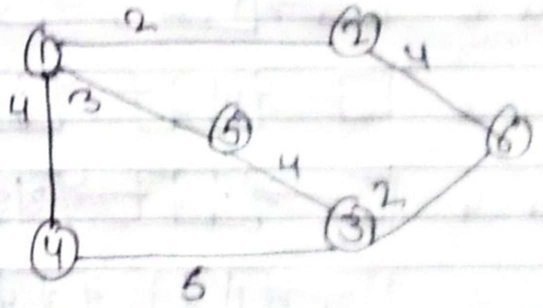
To represt an undirected graph with n vertices and e edges, it requires n head nodes and 2e list nodes.

(Ques'(b)

ans- Weighted - Graph :-

A graph is said to be weighed graph if every edge in the graph is assigned some weightage. It is denoted by w(e).

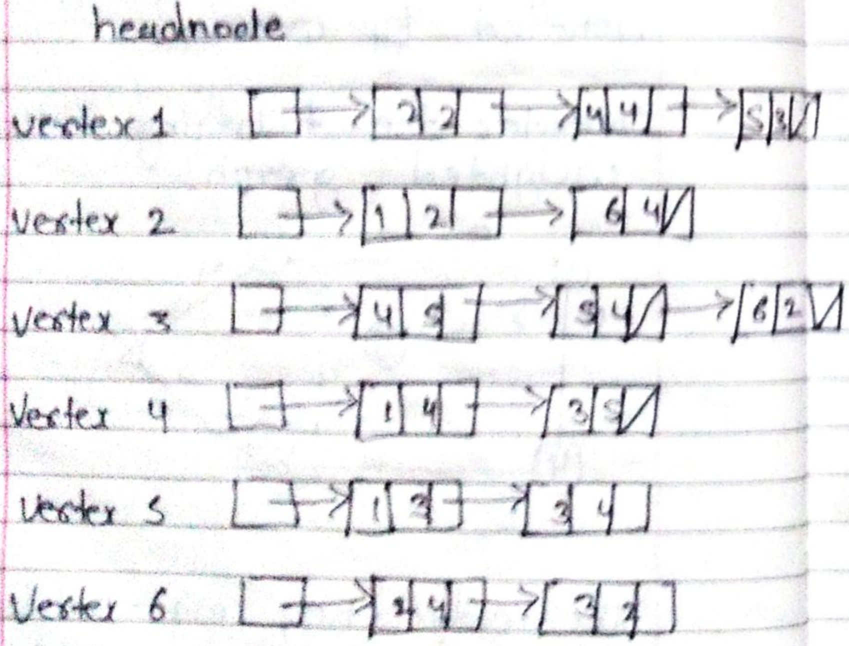
Consider the following undirected weighted graph!



The adjecency matrix for the above weighted graph is!

	1	2	3	4	5	6
1	0	2	0	4	3	0
2	2	0	0	0	0	4
3	0	0	0	5	4	2
4	4	0	5	0	0	0
5	3	0	4	0	0	0
6	0	4	2	0	0	0

The adjacency lis for the abov weighted graph is:-



Qus! (c)

ans	Tree	Graph
1	A tree is a finite nonempty set of nodes such that there is a special node called the root of the tree and all other nodes, if any, are divided into disjoint subsets of the root.	1. A graph G is defined as that which consists of two sets V and E where V is finite non empty set of vertices and E is set of pairs of vertices called edges.
2.	There are no loops in a tree	2 Graphs can have loops
3.	In tree there are numerous rules spelling out how connections of nodes can occur.	Graph has no rules dictating the connection among the nodes
4.	In tree there is an unique node called as root from which the subtree.	In the graph no such root node is there

Thara

Traversing in Binary tree:-

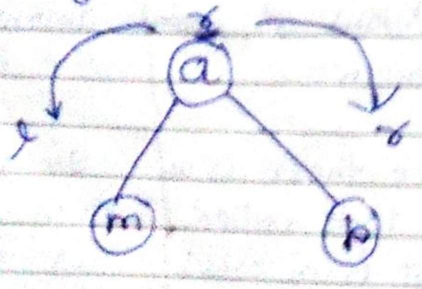
• Traversing in Binary tree can be done using the three ways:-

① Preorder
② Inorder
③ Post Order

① Preorder:- (r, L, R)

In traversing of preorder, following steps can be taken

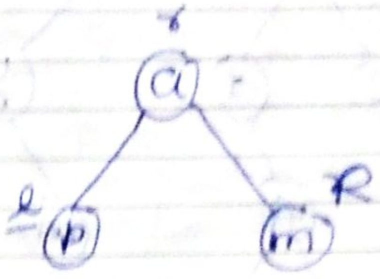
ⓐ Visit the root first.
ⓑ Traversing the left subtree.
ⓒ After step b, traversing in the right subtree.



Preorder :- a, m, p

Inorder :- (L, r, R)

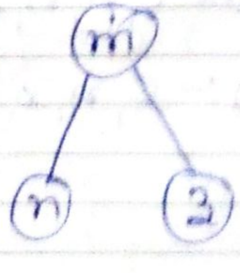
ⓐ Traversing in left subtree
ⓑ Visit the root node
ⓒ Traversing in the right subtree.



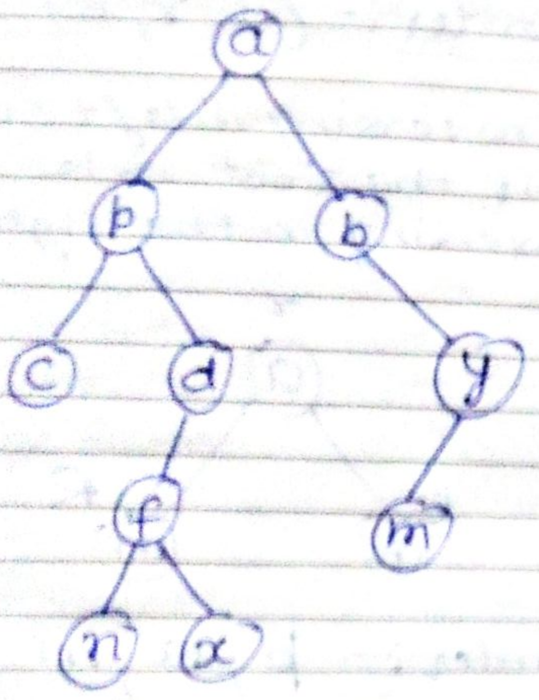
Inorder :- p, a, m

Postorder :- (L, R, r)

ⓐ Traversing to the left subtree
② Traversing to the right "
③ Visit the root node.



Postorder = n, 3, m



Traversing the above tree :-

Preorder (r,L,R) = a, p, c d, f, n, x, b, y, m

Inorder (L,r,R) :- c, p, n, f x, d, a, b, m, y

Postorder (L,R,r) :- c, n, x, f, d, p, m, y, b, a

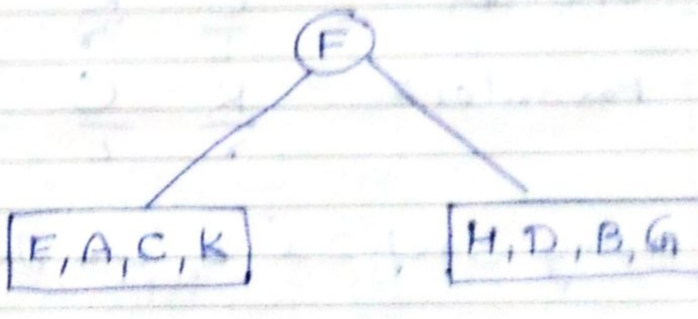
⇒ Draw a binary tree

Inorder :- E A C K F H D B G
Preorder :- F A E K C D H G B

Step-I (r,L,R) Preorder :- $\underset{r}{\underline{F}}$ $\underbrace{A\ E\ K\ C}_{L}$ $\underbrace{D\ H\ G\ B}_{R}$

Inorder = $\underbrace{E\ A\ C\ K}_{L}$ $\underset{r}{\underline{F}}$ $\underbrace{H\ D\ B\ G}_{R}$

using step I, we get

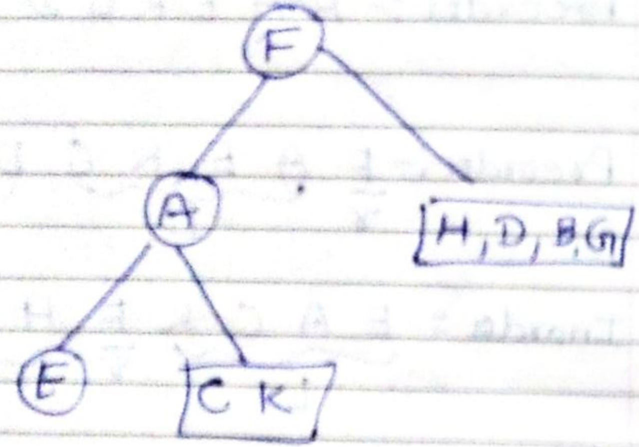


Step 2 Take the left subtree, from Step - I,

Inorder :-(L,r,R) $\underbrace{E}_{L}$ $\underset{r}{\underline{A}}$ $\underbrace{C\ \ K}_{R}$

Preorder :-(r,L,R) $\underset{r}{\underline{A}}$ $\underset{L}{\underline{E}}$ $\underbrace{K\ \ C}_{R}$

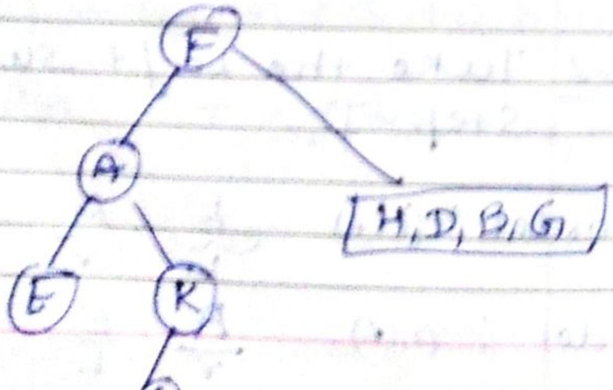
From step, 2, we construct



Step-3 Inorder = $\underset{L}{\underline{C}}$ $\underset{r}{\underline{K}}$ $\underset{R}{\underline{\quad}}$

Preorder = $\underset{r}{\underline{K}}$ $\underset{L}{\underline{C}}$ $\underset{R}{\underline{\quad}}$

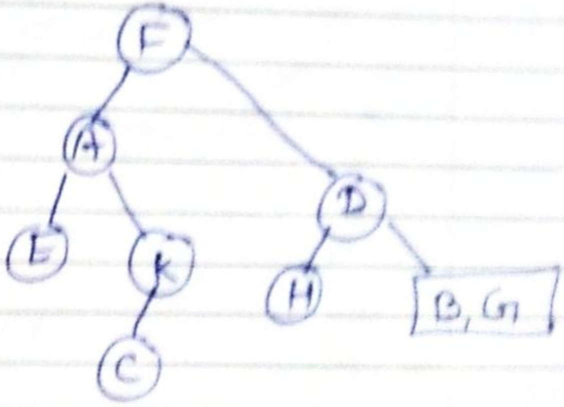
using step 2 & 3, we get



Step-4 Inorder = $\underbrace{H}_{L}$ $\underset{r}{D}$ $\underbrace{B\ G}_{R}$

Preorder = $\underset{r}{D}$ $\underbrace{H}_{L}$ $\underbrace{G\ B}_{R}$

using 3, & 4,,



Step5 Inorder:- $\underbrace{B}_{L}$ $\underset{r}{\underline{G}}$ $\underset{R}{\underline{\quad}}$

Preorder:- $\underset{r}{\underline{G}}$ $\underset{L}{\underline{B}}$ $\underset{R}{\underline{\quad}}$