# → Data structure ←

Data ⇒ Data is collection of information But it is in rawform. When Data is processed it becomes an information.

Data structure ⇒ It is the way of organising all data items that consider not only the element but also their relationship to each other.

"or"

Data structure is representation of logical relationship existing b/w individual elements of data.

"or"

Data structure is also defined as mathematical and logical model of particular organisation of data items.

"or"

Data structure is a building block of program.

# Classification of Data structure

Scanned by CamScanner

# www.askbooks.net

•AKTU Quantums •Toppers Notes •Books
•Practical Files •Projects •IITJEE Books

www.askbooks.net

All AKTU QUANTUMS are available

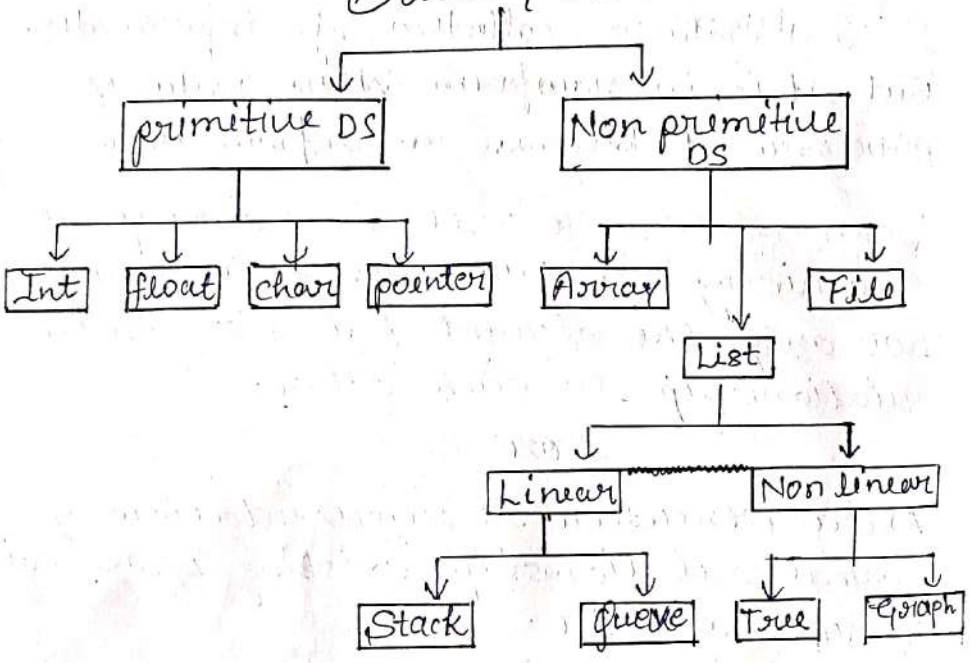
askbooks.net does not own the materials neither created it nor scanned it. We provide the links to the materials which are already available on the internet.

• Your complete engineering solution.
• Hub of educational books.

If you can buy the books then please do buy it, this website is solely dedicated to the underprivileged students who are willing to study but are short of resources.

1. All the ebooks, study materials, notes available on this website are submitted by readers you can also donate ebooks/study materials.
2. We don't intend to infringe any copyrighted material.
3. If you have any issues with any material on this website you can kindly report us, we will remove it asap.
4. All the logos, trademarks belong to their respective owners.

# Data Structure




**primitive DS →** These are basic DS and directly operated upon machine instructions. These have different representation on different computers.

Ex → Int, Float, char, pointer.

**Non primitive →** Non primitive DS are derived from primitive data structure. These are based on structuring of a group of homogeneous (same type) or heterogeneous Data items.

Ex → Array, file, List etc.

**List (linear link list) →** List is collection of variable no. of data items. And element of list must contain two fields.

(i) Data field (storing of Data)
(ii) Address field ( " " Address)

One for storing Data (Data field) and one for storing address (address field).

**File →** It deals with retrival of data storage and in computer programing language. It allows user to wright program to axes and use the data in the file.

## Linear →

**Stack →** It is an order list in which addition of new data items and division of already existing items is done from only one end, know as top of stack.

Stacks is basically works on LIFO.

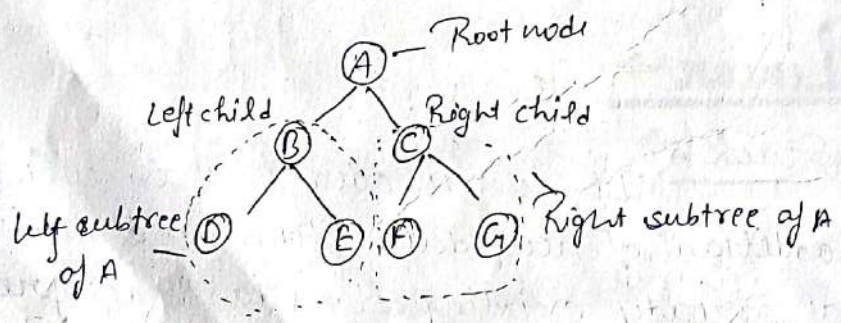
L I F O
last in first out

**Queue →** It is a non primitive linear data structure. It is an homogenous collection of element in which new elements are added in ~~which~~ one end called the 'Rear End' and the existing elements are deleted from other end called the 'Front end'

Queue is basically works on FIFO

F I F O
first in first out.

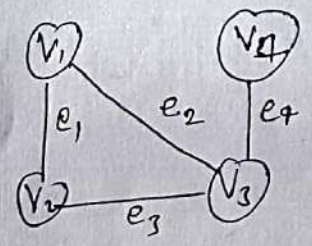
**Tree ⇒** It is non linear data structure. In which items are arranged in shorted sequence. It is used to reprisent hierarchical relationship among several data items.




**Graph :** A Graph 'G' consist of vertices 'V' and a set of edges (order) 'E'. It is denoted by G (V, E).

$V = \{V_1, V_2, V_3, V_4\}$

$E = \{e_1, e_2, e_3, e_4\}$




G(V, E)

**Time and complexcity →** It is the amount of time taken by the program for execution. Time complexcity is measur by using the frequency count.

```
void Display ( )
{ int a, b, c ;
  a = 10 ;
  b = 20 ;
  c = a+b ;
  P.F ( "%d", c) ;
}
```

| Code | freq. Count |
|---|---|
| a = 10 | 1 |
| b = 20 | 1 |
| c = a+b | 1 |
| P.f ("%d", c) | 1 |
| Total | 4 |

Scanned by CamScanner

```
int sum [int a [10], int n]
  {
  int i, sum = 0
  for (i=0, i<n, i++)
    {
    sum = sum + a[i];
    }
  return sum();
  }
```

| Code | Freq. Count |
|---|---|
| sum=0 | 1 |
| for (i=0, i<n, i++) | n+1 |
| sum = sum + a[i] | n |
| return sum | 1 |
| Total | 2n+3 |

Time comp

## Space complexcity →

The space complexcity can be defined as amount of memory/space required by an algorithm to run.

- Constant
- Instance ch

To compute the complexcity we use two factors.

Space requirment — $S_{(P)} = C + S_P$

Constant  Instance char

Algo.

```
Sum (a,n)
  S = 0.0;
  for i = 1 to n do
    S = s + a[i];
  return S;
  }
```

| Code | Space Count |
|---|---|
| S = 0.0 | O(1) |
| for i = 1 to n | O(n) |
| S = s + a[i] | O(n) |
| return s | O(1) |
| S.C | O(n) |

## Asymptotic Notations :-

Asymptotic notations is a shorted cody to represent the time complexcity.

Various notations such as 'O', 'Ω' & 'Θ' are used for calculating time complexcity.

## Big oh notation :-

(i) It is denoted by 'O'

(ii) It is a method of representing the upper bound of algorithm's running time.

(iii) Using big oh notation we can give longest amount of time taken by the algorithm to complete.

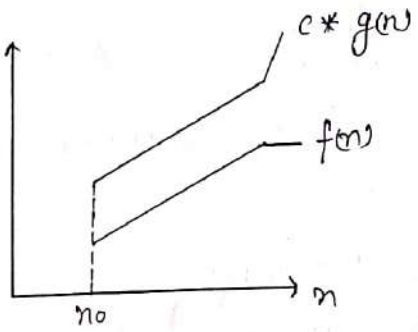
**Definition →** Let f(n) & g(n) be 2 non-ve functio

Let $n_0$ & c are two integers such that $n_0$

Scanned by CamScanner

denotes some value of I/P & $n > n_0$.
similary c is some constant such that ($c > 0$). We can write -

$$\boxed{f(n) \le c * g(n)}$$




$f(n) \in O(g(n))$

f(n) is less than g(n) is multiple of some constant (c).

Ex ⇒ Consider funciton $f(n) = 2n+2$ , $g(n) = n^2$ then we have to find some constant c so that $f(n) \le c * g(n)$

$f(n) = 2n+2$ , $g(n) = n^2$

for $n = 1$

$f(1) = 4$ ; $g(n) = 1$

$\boxed{f(n) > g(n)}$

for $n = 2$

$f(2) = 6$ , $g(n) = 4$

$\boxed{f(n) > g(n)}$

for $n = 3$

$f(3) = 8$ ; $g(n) = 9$

$\boxed{f(n) < g(n)}$

for $n > 2$ we obtain $\boxed{f(n) < g(n)}$

Thus always upper bound of existing time is obtain by 'O' notation.

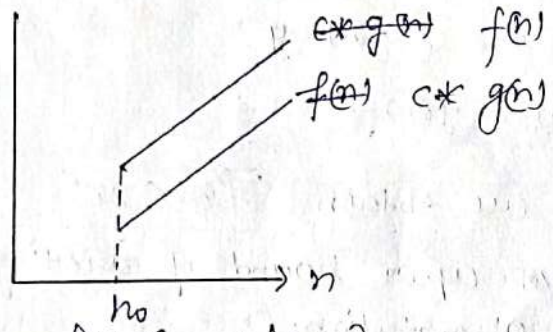
* For calculating the complexcity of an algorithom there are three case -
  (i) Best case.
  (ii) Worst case.
  (iii) Average case.

Omega notain ⇒
(i) It is denoted by '$\Omega$'.
(ii) It is used to represent lower bound of algarithem running time.
(iii) Using omega notation we can denote shortest amount of time taken by algorithem.

Definition ⇒ A function f(n) is said to be in $\Omega(g(n))$ if f(n) is bounded below by some positive constant is multiple of g(n) such that

$\boxed{f(n) \ge c * g(n)}$ for all $n \ge n_0$

Scanned by CamScanner




$f(n) \in \Omega(g(n))$

Example ⇒

$f(n) = 2n^2 + 5$

$g(n) = 7n$

for $n = 1$

$f(n) = 7$ , $g(n) = 7$

for $n = 2$

$f(n) = 13$ $g(n) = 14$

for $n = 3$

$f(n) = 23$ , $g(n) = 21$

$\boxed{f(n) > g(n)}$

Theta notation ⇒

(i) It is denoted by 'θ'.

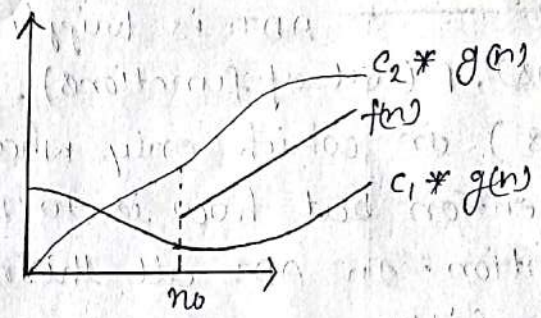
(ii) By this method the running time b/w upper bound and lower bound

(iii) Using 'θ' notation we can denote average ~~time~~ amount of time.

Definition ⇒ Let $f(n)$ & $g(n)$ be two non +ive functions. There are two +ive constants $c_1$ & $c_2$, such that $c_1 * g(n) \le f(n) \le c_2 * g(n)$

Then we can say that $f(n) \in \theta(g(n))$

$c_1 * (g(n)) \le f(n) \le c_2 * g(n)$




Example ⇒ $f(n) = 2n + 8$

$g(n) = 7n$

for $n = 1$

$f(1) = 10$ $g(1) = 7$

$f(n) = 2$

$f(2) = 12$ $g(2) = 14$

Scanned by CamScanner

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < n^4 \cdots < n^n$

lower — $\Omega$ | avg. — $\Theta$ | upper bound — Big-oh notation ($O$)

**Abstract Data type (ADT) ⇒** ADT is tripple of D (set of domains), F (set of functions), A (set of Axioms) in which only what is to be done is mention but how is to be done is not mention. In ADT all the implimentation details are ~~given~~ hidden.

| ADT = Type + function names + behaviour each function |
|---|

**ADT Operations ⇒** Various ADT operations are used in data structure.

Create ⇒ It creates data base.

Display ⇒ This operation is for displaying all the elements of data structure.

Insertion ⇒ By this operation

Deletion ⇒ By this operation any desired element can be deleted from the data structure.

Modification ⇒ This operation modified the desired elements value by any desired new value.

Advantages of ADT ⇒

(i) It is useful for representing all operations of the data structure.

(ii) The implimentation details can be hidden in ADT.

(iii) The implimentation details can be changed latter on during mentionance and bug fixing.

(iv) It helps to define the program definition in general form.

Difference b/w Linear & Non linear ⇒

| Linear DS | Non linear DS |
|---|---|
| (1) The elements are arranged in Linear order. | 1. The elements are arrang in non-linear order |
| 2. Simple to implement | complex to implement |
| 3. for ex ⇒ Stack, Queue | (3) Ex ⇒ Tree & graph |
| **Arrays ⇒** | |

Scanned by CamScanner

• Difference b/w Data Structure & Data type ⇒

| Data Structure | Data type |
|---|---|
| 1. It is an abstract description of a way of organizing data to allow certain operations on it performed efficiency. | 1. It represent basically type of data |
| 2. It can be reduced. | 2. It can't be reduced. |
| 3. Data structure is formed using data types. | 3. Data type are the primitive types. |
| 4. Ex ⇒ Array, linklist, Stack, Queue etc. | 4. Ex ⇒ int, char, float, double, string etc |
| 5. It can be a molecule. | 5. It can be an atom. |

# Array

**Array** ⇒ An array is collection of elements of similar data type.

'OR'

An array contains only one type of data.

'OR'

Array is collection of finite set of data items and these data items are stored adjecent memory location.

Syntax ⇒

Data type array name [size]

Ex ⇒ Int A[10]

Representations ⇒ 10, 20, 30, 40, 50

| | | | | |
|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 |
| a[0] | a[1] | a[2] | a[3] | a[4] |

Advantages ⇒

(i) Elements can be retrieved or stored very efficiently in sequencial organisation with the help of index or memory locations.

(ii) All the elements are stored at continuous memory location.

Disadvantages ⇒

(i) Insertion and deletion of elements is complicated due to sequencial nature.

(ii) for storing the data large continuous free block of memory is required.

Scanned by CamScanner

(Left & right & null parts)

(iii) memory frugmentation occurs if we remove the elements randomly.

## Types of array →

(i) Single dimensional
(ii) Multi Dimensional.

Row ~~measur and Co~~

**Row major and Coloum major as Address calculation of an array** → When a two-D array stored in computers memory all the elements of array get stored at the successive memory location. And this location of elements is parte on 2-type representation.

~~Row major representation~~

**One Dimensional array** → One-D array is one in which only one subscript specification is needed to specify a particular element by the array. 1-D array can be declared as follows

data type - variable name [subscript]

int - a [5]

↳ no of values stored in the array

(i) must be an int value.

| Index | element (2, 3, 4, 5, 7) |
|---|---|
| a[0] | 2 |
| a[1] | 3 |
| a[2] | 4 |
| a[3] | 5 |
| a[4] | 7 |

Here we can say that first element have the subscript is zero. To axes a particular element in the array.

Size of the array can be determine as follows.

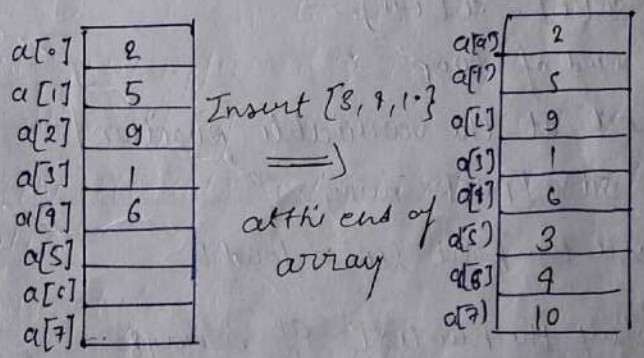
$$\boxed{\text{size} = \text{upper bound} - \text{lower bound} + 1}$$

$$= 5 - 0 + 1$$
$$= 6$$

Insertion in 1-D array ⇒

(i) Insert at the end of array ⇒ Inserting an element at end of the array can be easily done by providing the remaining memory space.

Ex ⇒ In the array a[8] = {2, 5, 9, 1, 6}.
insert = {3, 4, 10}

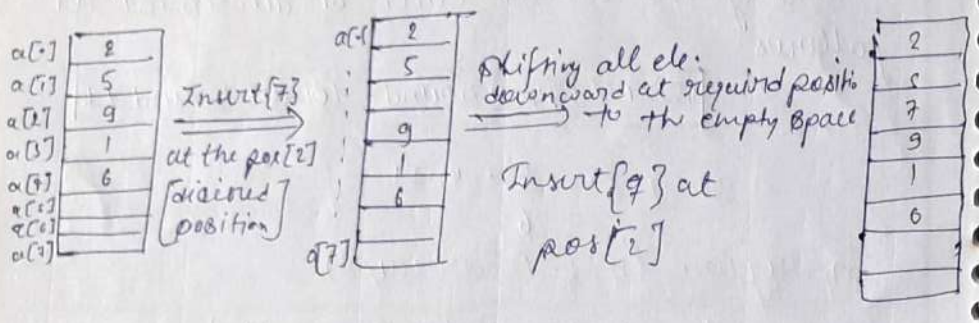



(ii) Insert at desired position ⇒ Inserting the elements its required position elements must be moved down words to new locations. To

Scanned by CamScanner

accomdate and keep the order of elements.

Ex In the array a[8] = {2, 5, 9, 1, 6}, insert {7} at pos[2].




Algo (a, len, pos, num)

Linear array — to no. of elem — position at which num is inserted

1. Intialize the value of i [set i = len]
2. Repeat for i = len down to pos
   {shift the ele down to 1 position}
   Set a[i] = a[i+1]
   end of loop
3. [Insert at the varriable position]
   Set a[pos] = num;
4. [Reset len] Set len = len+1
5. Display the new list of arrays.

End

## Deletion in one dimensional array ⇒

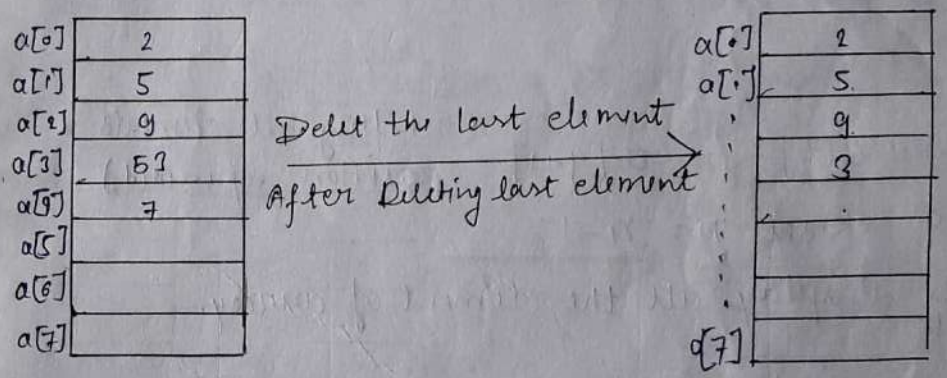
(i) Deletion at the end of array.
(ii) Deletion at desired position.

### (i) Deletion at the end of array ⇒

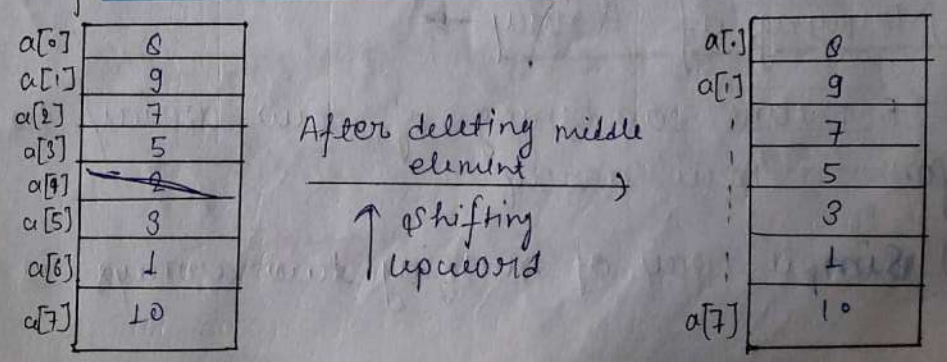
⇒ Deleting the one element of the end of array presents no difficulty, but deleting element somewhere in the middle of the array would required to shift all the elements to field the space blank by the deletion of the element.

⇒ Then the elements following its were moved upward by one location.

⇒ A = {2, 5, 9, 8, 7} delete the last element



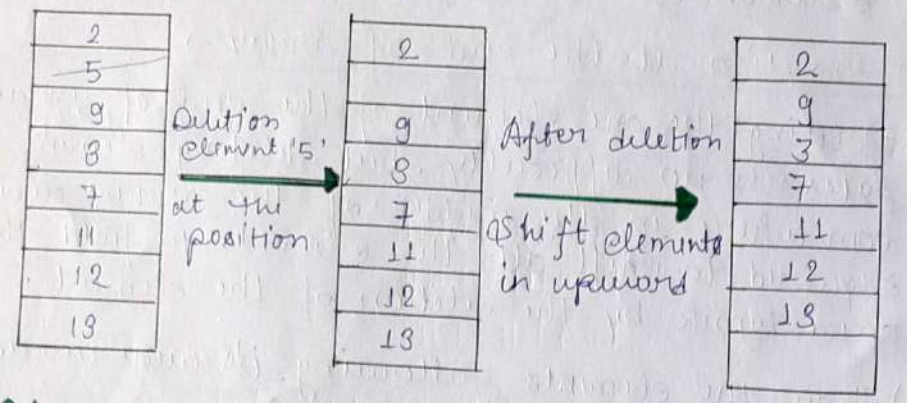

If delete middle element ⇒




Scanned by CamScanner

## Deletion at desired position →

a[8] = {2, 5, 9, 8, 7} Delete [5] at position 'a[1]'




## Algo.

1. get item = a [pos]
2. Repeat for x = position -1
3. a[j] = a [j+1] (shifting all elements 1 position upward)
4. Reset n = n-1;
5. Display all the element of array.

End

## Merging of Array →

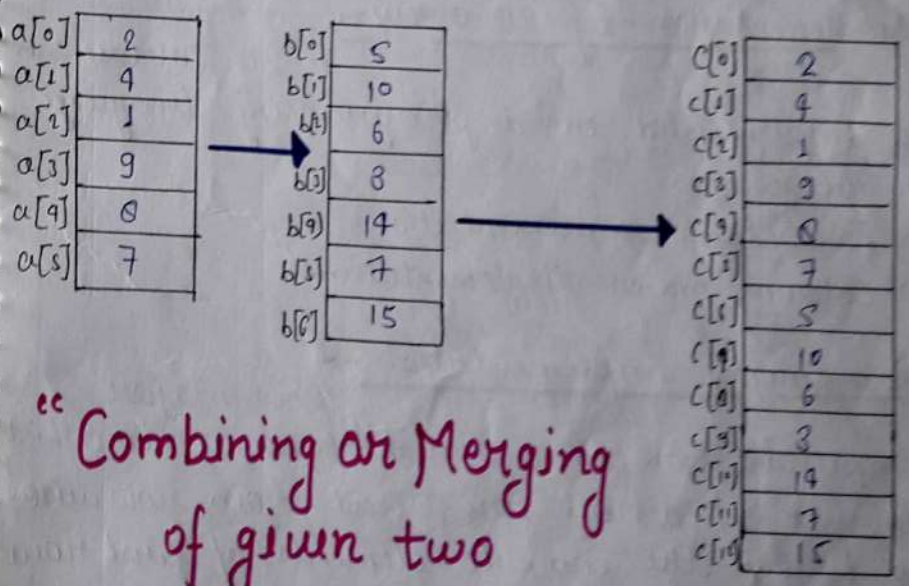
⇒ It means combining of two array to form a new array.

⇒ Simple way of merging two arrays is that first copy all elements of one array into a third empty array and then copy all the elements of other array into third array.

→ The elements of second array copied into third array after the position. first array ~~after~~ are copied.

→ Another way is short the array during merging.

**Example →** If we find some element in two array called dupli. And its remove by shorted sequence.




"Combining or Merging of given two array"

Scanned by CamScanner

## Two dimensional array ⇒

2-D arrays are grid for a total no. of hundred elements. The first no. in brackets is the no. of rows, the second no. in the brackets is the no. of coloums.

| | | |
|---|---|---|
| [0][0] | [0][1] | [0][2] |
| [1][0] | [1][1] | [1][2] |
| [2][0] | [2][1] | [2][2] |

int array 2D [10] [10]

↑ Data type ↑ variable name ↑ Row ↑ Coloum

## Implementation of 2D array ⇒

A 2D array can be implemented in a programing language in two ways.

(i) Row Major implementation.

(ii) Coloum major implementation.

## Row major implementation →

Row major implementation is a linear technic in which elements of array are used from row wise. i.e firstly the complete elements of first row is stored in memory then the complete second row is stored and so on.

int array [3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9}

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} a[3][3] = \boxed{a_{00} \mid a_{01} \mid a_{02} \mid a_{10} \mid a_{11} \mid a_{12} \mid a_{20} \mid a_{21} \mid a_{22}}$$

## Adress of calculation of elements for row major implimintation →

The computer doesn't keep the track of all elements of array, rather it keep a base address (i.e the adress of first element in the array) and calculated the address of required element when needed. It calculate this address in row major by the following formula.

$$\boxed{\text{Adress of } a[i][j] = B + W[n(i-L_1) + (j-L_2)]}$$

B ⇒ Base address

W ⇒ Storage space

i = Row element

$L_1$ = Lower bound of ~~element~~ row

j = coloumn element

$L_2$ ⇒ upper bound of column

n → no. of columns

Where; $n = U_2 - L_2 + 1$

$U_2$ → upper bound of column; $L_2$ → lower bound of row

add. of $a[i][j] = B + i * n + j$

Scanned by CamScanner

Where

Array is in form of $a[m][n]$

size of rows & columns

Q. int $a[2][3] = \{10, 20, 30\}$ we find of add. of $a[1][2]$

$B = 102$

Sol$^n$ add. of $a[1][2] = B + i*n + j$

$= 102 + 2 \times 2 + 3$

$= 102 + 4 + 3$

$= 109$

Q. A 2D array is defined as ~~for~~ $[4, 7, -1 : 3]$ required two bytes of storage space each. With the array stored in row major form. Then calculate the address of element at location $a[6, 2]$ if the BA is 100.

Sol$^n$

$B = 100$

$N = 2$ bytes

Lower bound of ~~array~~ row $L_1 = 4$

Lower " " column $L_2 = -1$

upper bound of row $U_1 = 7$

" " " column $= 3$

Row no. of required elements $= 6$

Column " " " " $j = 2$

$n = U_2 - L_2 + 1$

$= 3 - (-1) + 1$

$n = 5$

$B + N[n(i-L_1) + j - L_2]$

Address of $a[6][2] = 100 + 2[5(6-4) + 2 - (-1)]$

$= 100 + 2[10 + 3]$

$= 126$

Q. Consider the linear array $A[5:50]$, whose ~~add~~ $B = 300$ and the no. of words per memory cell is 4. Find the address of $A[15]$.

$B = 300$

$i = 15$

lower bound of row $= 5$

lower " " column $= 50$

$A[15] = 300 + [15 - 5] * 4 \Rightarrow B + (i - L_1) \times$

$= 300 + 40$

$= 340$

Given

$A[r_l \ldots r_u, c_l \ldots c_u]$

$r_l$ = lower of row

$r_u$ = upper bound " row

BA = 1 $\quad$ C = size of an element

RMO:-

No. of row, $NR = r_u - r_l + 1$ (No. of elements in 1 column is equal to no. of rows)

No. of column, $NC = C_u - C_l + 1$ (No. of " " in 1 row is equal to no. of columns)

$loc\ A[i][j] = BA + [(i - r_l) \times NC + (j - C_l)] \times C$

Scanned by CamScanner

(12) A [-100 ... 100 , -250 ... 250]

C = 2  BA = 1000  NR = ru - rl + 1 = 100 + 100 + 1 = 201

loc A[4][4] = ?  NC = 250 + 250 + 1 = 501

A[4][4] = 1000 + [(4 + 100) x

= BA + [(i - rl) x NC + (j - cl)] x C

= 1000 + [(4 + 100) x 501 + (4 + 250)] x 2

= 105716

= 1000 + [(4 + 100) x 201 + (4 + 250)] x 2

= 93316

(2) Given a 2D array A [-100 : 100, -5 : 50] (rl, ru, cl, cu). Find the address of element A [99, 49] = ? considering the base add BA = 10. Each element required 4 bites for storage.

NC = 50 + 5 + 1

= 56

A[99, 49] = 10 + [(99 + 100) x 56 + (49 + 5)] x 4

= 44802

(3) x [20, 50] (m, n)  Row major

C = 4

BA = 200

x [10, 10] = ?

For a[m][n] array we can locate address of a[i][j] using following formula.

Row major a[i][j] = BA + (i x n + j) x C

= 2000 + [10 x 50 + 10] x 4

= 4040

Column major = BA + (j x m + i) x C

= 2000 + (10 x 20 + 10) x 4

= 2840

(4) Consider a 2D array C[20, 30] (m, n). Element type is integer, if the BA = 1076 what will be the address of C[17][29]

loc C[17][29] = BA + [i x n + j] x C

Row major = 1076 + [17 x 30 + 29] x 2

= 2154

loc C[17][29] = BA + [j x m + i] x C

Column major = 1076 + [29 x 20 + 17] x 2

= 2270

Scanned by CamScanner

Column major order ⇒ Memory allocation is done is ~~by~~ column by column i.e first the elements of complete first column is stored then ~~complete~~ elements of complete second column is stored & so on.

Example.

$$A[3][4] = \begin{array}{c} \\ 0^{th} \\ 1^{st} \\ II^{nd} \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 00 & 01 & 02 & 03 \\ 10 & 11 & 12 & 13 \\ 20 & 21 & 22 & 23 \end{bmatrix} \end{array}$$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 10 | 20 | 01 | 11 | 21 | 02 | 12 | 22 | 03 | 13 | 23 |

← Colo1 → ← Colm 2 → ← Colm 3 → Col 4 →

Offset → Row by colum, Colm by Row

$$\boxed{LocA[i][j] = BA + [m(j - l_2) + (i - l_1)] \times W}$$

$L_2$ ⇒ lower bound of Col.
$L_1$ = lower " " Row.

or

$$\boxed{Loc\ A[j][k] = BA + [M(k-1) + (j-1)] \times N}$$

M = Total no. of rows.
J = row. no. of the element.
K = column no. of the element.

Qun ⇒ $A[\overset{l_1}{-20} \ldots 20, \overset{l_2}{10} \ldots 35]$
$l_1$ = lower row, $u_1$ = upper row, $l_2$ = lower col, $u_2$ = upper col

N = 1 byte
BA = 500

$M = 20 + 20 + 1 = 41$

Loc A[0][30].

$Loc\ A[0][30] = BA + [m(j - l_2) + (i - l_1)] \times W$

$= 500 + [41(30 - 10) + (0 + 20)] \times 1$

$= 1390$

(2) Consider a 2D array $A[\overset{l_1}{-90} \ldots 90, \overset{l_2}{-10} \ldots 40]$ element type is integer find the address of A[89][39] BA = 10 and each element required 4 bites.

$m = 90 + 90 + 1 = 181$

$Loc\ A[89][39] = BA + [m(j - l_2) + (i - l_1)] \times w$

$= 10 + [181(39 + 10) + (89 + 90)] \times 4$

$= 36202$

(3) Consider the linear arrays AAA[5:50] BBB[-5:10] & CCC[1:8] find the elements in each array.

No. of elements = upper bound − lower bound + 1

for array AAA $= 50 - 5 + 1 = 46$

BBB $= 10 + 5 + 1 = 16$

CCC $= 8 - 1 + 1 = 8$

Scanned by CamScanner

Suppose Base [AAA] = 300 & W = 4 memory cell for AAA. Find the address of AAA[15], AAA[35].

Location AAA[35] = BA + co × [index − lowerbound]
= 300 + 4 × [35 − 5]
= 420

AAA[15] = 300 + 4 [15 − 5]
= 300 + 40
= 340

AAA[55] = 300 + 4 [55 − 5]
= 300 + 200
= 500

The AAA[55] is not possible as upper limit is 55.

int [i][j]

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1000<br>10 | 1002<br>20 | 1004<br>30 | 1006<br>40 |
| 1 | 1008<br>50 | 1010<br>60 | 1012<br>70 | 1014<br>80 |
| 2 | 1016<br>90 | 1018<br>100 | 1020<br>110 | 1022<br>120 |

*A]
*(*A)

| 1000 | 1002 | 1004 | 1006 | 1008 | 1010 | 1012 | 1014 | 1016 | 1018 | 1020 | 1022 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |

← $R_1$ → ← $R_2$ → ← $R_3$ →

If we find the value of array.

A[i][j] = *[*(A+i)+j]

* → shows only address
** → shows value.

Ex ⇒

A → 1000
or
A+0 → 1000
A+1 → 1008
A+2 → 1016
(A+1)+1 ⇒ A+2

* A = 1000
*[A+0] ⇒ * A[0] = 1000
*[A+1] → * A[1] = 1008
*[A+2] = 1016

Application of Arrays ⇒

Order list ⇒ It can be represented using arrays.

(i) The order list is useful for polynomial rep^n, addition & multiplication of two polynomials.

(ii) The arrays are also use for representing some sequencial data structure.

(iii) Arrays are used to represent sparse matrishes.

Sparse Matrices & their Representations ⇒

As parse matrix is that matrix which has few non zero

elements as compare to size (M×N) of the matrix.

Ex⇒ A array (M×N) (100×100) has 10 element are non zero. Accesing this 10 element on this to make 10000 times scan. Also only 10 spaces will be with non zero elements & remaining spaces of matrix will be filled with zero only.

Types of sparse matrix ⇒

It is basically two types.

① Upper triangual ~~mat~~ sparse matrix ⇒

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 0 | 6 | 7 | 8 | 9 |
| 0 | 0 | 10 | 11 | 12 |
| 0 | 0 | 0 | 13 | 14 |
| 0 | 0 | 0 | 0 | 15 |

Upper triangular matrix

A matrix is said to be upper sparse matrix its non zero elements are found only in the upper triangle of the matrix including the main diagnal.



Lower triangular sparse matrix ⇒ A matrix is said to be lower triangular sparse matrix if its non zero elements are found only in the lower triangular of the matrix including the main diagnal.

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 3 | 0 | 0 | 0 |
| 4 | 5 | 6 | 0 | 0 |
| 7 | 8 | 9 | 10 | 0 |
| 11 | 12 | 13 | 14 | 15 |

Representation of sparse matrix

The representation of sparse matrix will be a triplet only. In the cells that the sparse matrix means very few non-zero elements having in it. Rest of the space are having the values zero. which are basically unless values or empty values. So in this efficient representation we will consider all the non-zero values along with their positions.

In a row wise representation of sparse matrix (2-D matrix). The zeroth (0th) row will stored total row of the matrix, total column of the matrix & total non-zero values

Scanned by CamScanner

Total no. of rows — Total no. of column — Total no. of non zero ele.

| Index | Row no. | column no. | value |
|---|---|---|---|
| 0 | 6 | 7 | 8 |
| 1 | 0 | 6 | -10 |
| 2 | 1 | 0 | 55 |
| 3 | 2 | 5 | -23 |
| 4 | 3 | 1 | 67 |
| 5 | 4 | 6 | 88 |
| 6 | 5 | 4 | 14 |
| 7 | 6 | 4 | -28 |
| 8 | 7 | 5 | 99 |

memory space ⇒ (Total no. of non zero value+1) × 3 × 2 size

$= (8+1) 3 \times 2$

$= 54$

~~Unsaved space~~ =

unused space = 84 − 54

= 30

## Operations of arrays ⇒

1. Insertion of element 'u' at its desired position in the array.
2. Deletion. 'K'.
3. Find the key element from an array.

**Alg** Insert an element into the $k^{th}$ position

Insert (X, Y, K, item)

X — linear array; K — (+ve integer) $(K < Y)$

1. Set A = Y
   - move the $A^{th}$ element.

Column major = BA + w[mj+i]

Row major = BA + w[ni+j]

no. of elements in array = UB − LB + 1

Address of any array ~~= BA + w~~

A[K] = BA + w[K − LB]

Scanned by CamScanner

# LINK LIST

Link list ⇒ A link list is a set of nodes or it is a special list of some data or elements linked to one another.

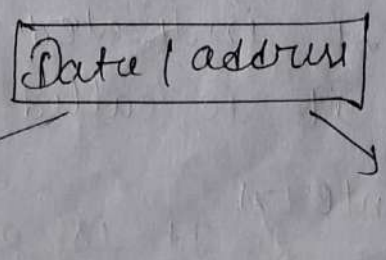
⇒ The logical ordering is represented by having each element pointing to the next element.

⇒ Each element is called a node and each node has two fields "data field" [Info] "Address field" [Link, pointer]

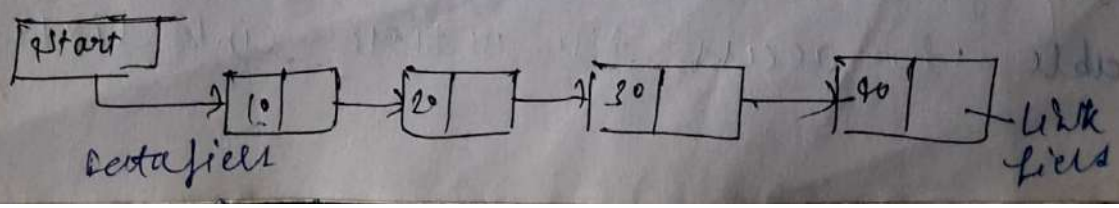
(i) Pointer field ⇒ Link field use to point to next node and store the address of next node Basically link field is nothing but address only.

(ii) Data field ⇒ The data field store actual piece of information.

⇒ Structure of Node ⇒




Ex ⇒ 10 20, 30, 40




Scanned by CamScanner

Advantages of link list ⇒

1. Link list are dynamic data structure.
2. Efficient memory utilization.
3. Insertion & Deletion are easier and efficient
4. many complex application can be easily carried out with link list.

Disadvantages of link list ⇒

1. It require more memory if the no. of field are more
2. Access to a arbitary Data item is little bit complex and also time consuming.

Link list termenology ⇒

(i) Data field.
(ii) Address field [Link field]
(iii) Null pointer ⇒ The link field of the last node contains null rather than avalid address. It indicated the end of the list so that it is called null pointer.
(iv) External Pointer ⇒ It is pointer to carry first node in the link list. It enable to access the entire link list.

(5) Empty list [Null list] ⇒ If the nodes are not present in the link list. Then it is called empty list or null list.

Value of external pointer will be ⇒ 0

Notation ⇒

'p' → pointer

Node (P) ⇒

Data (P) →

link (P) →

Operation of a link list ⇒

[1] Creation ⇒

This operation is use to create a link list.

[2] Insertion ⇒

This operation is use insert a new node in the link list at the specified position.

→ A new node may be inserted as -

Scanned by CamScanner

(i) At the beging of a link list.

(ii) At the end of a link list.

(iii) At the specified position in the linked list.

(iv) If the list is itself empty then the new node inserted as the first node.

(3) Deletion ⇒ This operation is use to delete an item from the link list.

⇒ A node may be deleted as.

(i) At the beging of a link list.

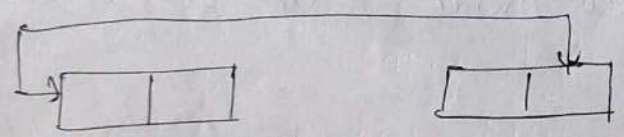
[2] End of a link list.

[3] At the specified position in the first list.

[4] Traversal ⇒ It is a process of going through all the nodes. All the nodes of link list from one into the other end.

Fv ⇒ Toward the last node.

Start



Called forward traversing

(5) Concatination ⇒ It is process of appending the second list to the end of the first list consisting of m node.

⇒ When we concatinate two list, the second list will be of having (m+n) nodes.

Display :- This operation is use to print each and every node information. This are the basic operation are performed in the link list.

## Types of link list →

There are four types of link list that are following -

(i) Singly link list → Singly linear L.L. / ~~doubly linear L.L~~ Singly circular L

(ii) Doubly link list → Doubly circular L.L.

(iii) Circular like list.

(iv) Circular doubly link list.

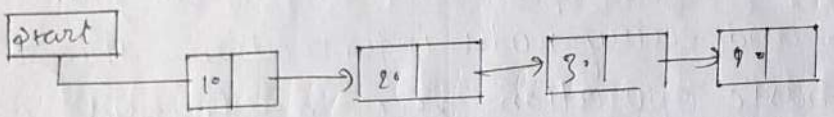
## Singly linked list ⇒

Singly linked list is o in which all nodes are linked together in subsequencial manner. It is called singly linear linked list because this list consis of only one link. To point to next node or

This is also called linear link list beca the last element points nothing. It is linear

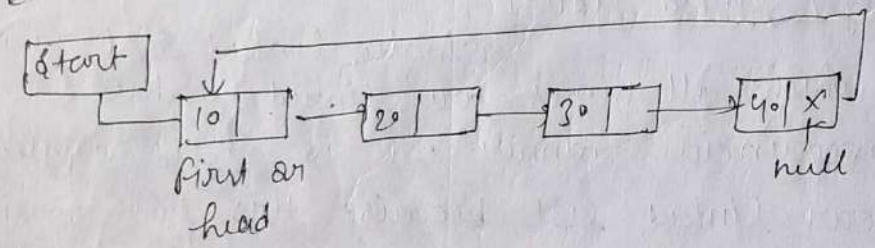
Scanned by CamScanner

nature. The last field of the last not is null which means that there is no further list. The problem with this list is that we can't access the predecessor of ~~node~~ from the current node.




Singly circular linked list ⇒ In this type of linked list only one ~~point~~ ~~to~~ link is use to point to next element and ~~its~~ this list is circular means that the last nodes link field point to the first or ~~node~~ head node. Last nodes's field point to the field.

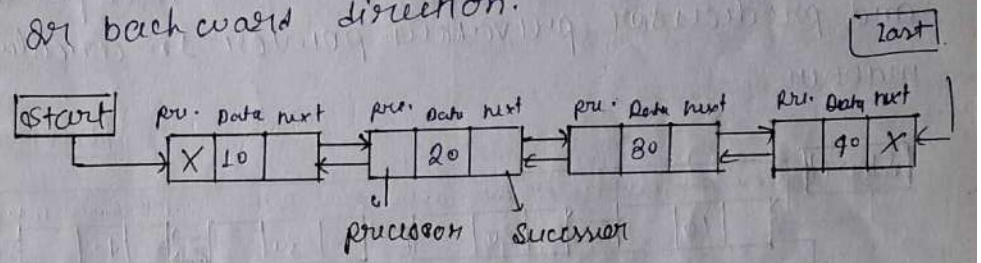
Ex ⇒



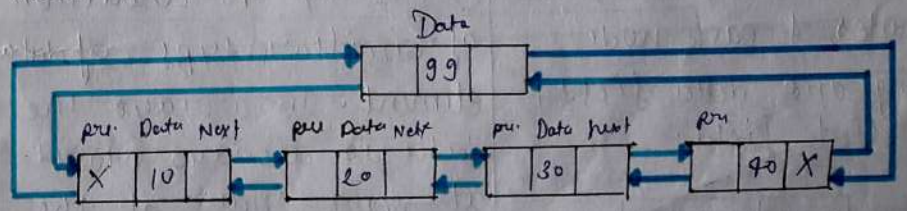

Doubly link list ⇒ This list is call because each node has two pointers or two nodes, successor node (next pointer) & predecessor node (previous pointer).

⇒ The previous pointer point to previous node & next pointer points to next node.

⇒ Only in case of head node the previous pointer is obiously null & last node's next pointer points to null. This help to traverse the list in the forward direction or backward direction.

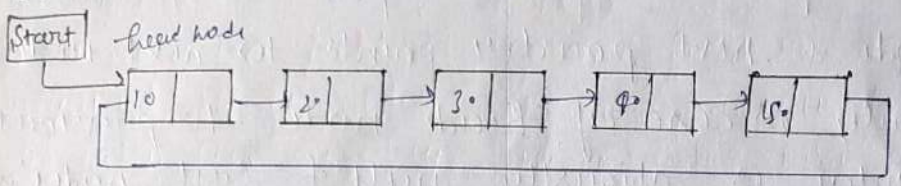



Doubly circular linked list ⇒ In circular doubly link list the previous pointer of first node & the next pointer of last node is pointed ~~to~~ head node.

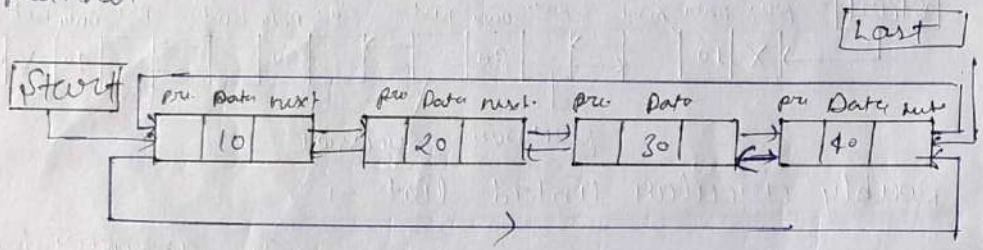



Circular linked list ⇒ It is one which as no beginning & no end it is similarly to linked list except the last nodes next pointer point to first node. A singly link list can be

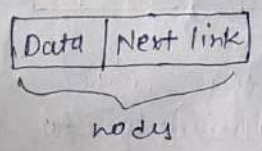
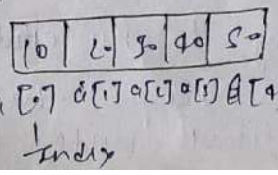
made a circular link list by simply shorting the address of the very first node in the link field of the last node.




Circular doubly link list ⇒ In which is one which has both the sucessor next pointer and predicessor previous pointer in circular manner

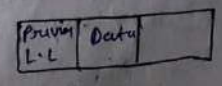



Difference b/w array & link list ⇒

| Link list | Array |
|---|---|
| ① The link list is collection of notes & each node having one data field & next link field.   | ① Array is collection of similiar type of data element. In arrays the data is always stored at some index of the array. Ex ⇒   |
| ② Any element can be access by sequntical access only. | ② Any element can be access randomly. i.e with the help of Index of the array |
| ③ Physical the data can be deleted. | ③ Only logical deletion of data is possible. |
| ④ Memory alocation is dynamic. Hence developer can allocate as well as dealocate the memory. So no wasteg of memory is there. | ④ Memory allocation is static. Hence once the fixed amount of size is declared then that must memory allocated. Therefore there is a chance of either memory wastage or memory shortage. |
| ⑤ Insertion & deletion of data is easy. | ⑤ Insertion & deletion of data is difficult. |

program of doubly & ~~singly~~. circular.

Difference b/w singly & doubly ⇒

| Singly Link list | Doubly link list. |
|---|---|
| ① Singly is collection of nodes and each node is having one data field & next field. Ex ⇒ | ① It is collection of nodes and each node is having one data field, one previous link field & next link field. Ex ⇒   |

Scanned by CamScanner

| | |
|---|---|
| ② The element can be ~~acc~~ access using next link field. | The element can be ~~acc~~ access using both previous link field as well as next link. |
| ③ No ~~st~~ extra field is required hence nodes can take less memory. | ⑧ One field is required to store previous link hence nodes take more memory. |
| ④ Less efficient access to elements. | ④ ~~Acc~~ More efficient to access to element. |

Creation of D.L.L. ⇒

1⇒ Intialize a variable flag = True. (The purpose of this flag is for making a check for first node)

| pre. | Data | Next |
|---|---|---|
| Null | 10 | Null → |

Now flag = false

2⇒ After creating first node reset flag.

3⇒ If head node is created we can further create a link list. By attaching the subsequent node. suppose we want to insert a node with value 20

| | |
|---|---|
| Null \| 10 \| Null | New / head / temp |
| head / temp | |
| Null \| 20 \| Null | |
| new | |

Binary search Tree

| | |
|---|---|
| Null \| 10 ⇄ 20 \| Null | temp → Next = new |
| head / temp, new | |
| Null \| 10 ⇄ 20 \| X | temp = New (/ Newtemp is moved a head*/) |
| head, New / temp | |

8) If user wants to insert a node with values 30.

| | |
|---|---|
| Null \| 10 \| ~~Next~~ ⇄ 20 ⇄ ~~X~~ Prev \| 30 \| X | |
| head, temp, New | |
| 10 ⇄ 20 ⇄ 30 \| X | |
| head, new / temp | |

temp = new

Dynamic implitentation ⇒ (or) Dynamic Memory allocation function ⇒ There are two types of memory management techic.

(i) Static memory Management.

(ii) Dynamic memory Managment.

Static memory management ⇒ The memory allocation & deallocation is done at compilation time.

Dynamic memory managment ⇒ The memory allocation & delocation is done at run time.

Scanned by CamScanner

Difference b/w dynamic memory & Static memory ⇒

| Static memory | Dynamic memory |
|---|---|
| 1. The memory allocation is done at compile time | 1. The memory allocation is done at run time. |
| 2. Prior to allocation of memory some fixed amount of memory must be decided. | 2. ~~Ho~~ Know need to no amount of memory prior to allocation. |
| 3. Wastage of memory & shortage of memory. | 3. Memory can be allocated as per requirement. |
| 4. It is less efficient than dynamic memory. | 4. It is more efficient |
| 5. Ex ⇒ Array | 5. Ex ⇒ Link list |

* Various function for dynamic memory allocation

① Malloc ② Calloc ③ Rulloc ④ Free

Malloc ⇒ The malloc func^n is use to allocate the memory space as per requirment.

* This function does not intitize the memory allocated during execution. Infact it carries garbage value.

* This function allocate requested block of memory and returns pointer of type (void*) to the start of that block.

* If function fails it returns null. Therefore it is necessary to verify if pointer return not null.

Malloc (size)
(* temp)

Malloc (size of (int))

Calloc ⇒

* The calloc similar to malloc but it intilize the allocated memory to 0

⇒ The calloc ( ). takes two arguments first is no. of elements & II^nd is type of element & compute no. of bytes to allocate

⇒ It allocates requested block and return pointer to void pointing to begin of block. If fails returns null.

~~ca~~ calloc (number, size);

Rulloc ⇒ Rulloc function is used to modify the size of allocated block by malloc & calloc.

Scanned by CamScanner

Malloc & Calloc() function to new size, If enough space does not exist in memory of current block to extend, new block is allocated for the full size of ~~p~~ reallocation then coppies the existing data to new block and then free the port block.

realloc (pointer name, size);

Free ⇒

The free function is used to deallocate the ~~de~~ allocated memory.

free (pointer name);

Difference b/w Malloc & Calloc ⇒

| Malloc () | Calloc () |
|---|---|
| 1. Malloc() stands for memory allocation. | 1. calloc() stands for charcter allocation. |
| 2. Malloc () takes one argument i.e no. of bytes | 2. Calloc () takes two arguments those are no. of blocks & ~~type~~ of each block |
| 3. Syntax of Malloc () void * malloc (size t, n); allocate n bytes of memory. If the allocation succeed, a void pointer to the allocated memory is return. otherwise null is return | 3. ~~void~~ syntax of calloc () void* calloc (size_t, n, size_t size); allocates a contious block of memory large enough to hold of n elements of size bytes each. The allocated reson is initilize is zero. |
| 4. Malloc() is faster than calloc. | 4. Calloc () takes longer time to execute than malloc |
| 5. The contains of allocated memory is grabage values | 5. The allocated memory is reson is initilize to zero |

(Application of link list ⇒

Addition of two polynomials ⇒

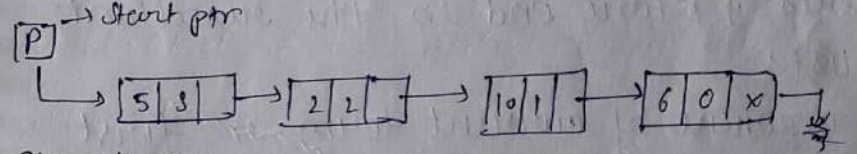
$P_1(x) = a_n x_n^e + a_{n-1} x_{n-1}^e + \cdots + a_1 x_1^e + a$

$P_2(x) = b_n x_n^e + b_{n-1} x_{n-1}^e + \cdots + b_1 x_1^e + b$

'Node' can be used to add 2 poly

Node → Coefficient field
→ Exponent field
→ Link field

$P_1(x) = 5x^3 + 2x^2 + 10x + 6$




Structure ⇒

```
struct poly
{int coff;
 int Expo;
struct poly * ptr → Self referential field
};
type def struct polynode PNODE;
```

Scanned by CamScanner

Algo

① Read the no. of terms in the first polynomial p.

② Read cofficient & exponent of first polynomial.

③ Read no. of terms in second polynomial Q.

④ Read cofficient & exponents of second polynomial.

⑤ set temp. ptr (p & q) to traverse two polynomials respectively.

⑥ Compare the exponent of two polynomials starting from first node.

(a) If both exponents are equal then add the cofficients & store end in the resultant link list.

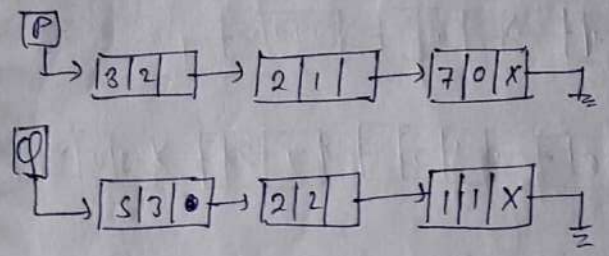
(b) If exponent of current term in first polynomial p < exp. of Q then add ~~coffici~~ term of q in resultant link list and move the pointer q to point next node in second polynomial.

(c) If exp. of (p) > exp. of (Q) then add p term in the resultant link list and increase p→p+1

(d) finally append the remaining node of either of the polynomials to the resultant link list.

Ex ⇒ $p = 8x^2 + 2x + 7$

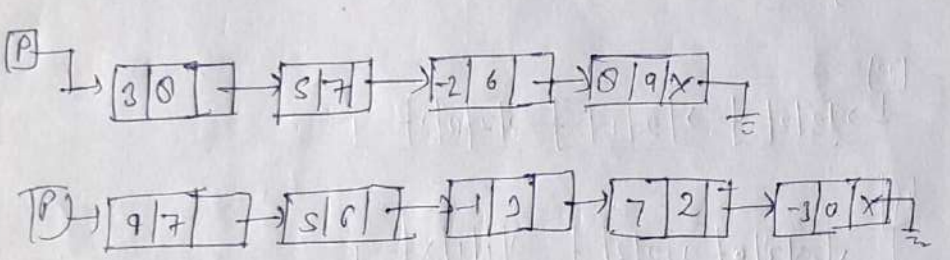
$Q = 5x^3 + 2x^2 + x$




Scanned by CamScanner

$p = 3x^8 + 5x^7 - 2x^6 + 8x^9$

$q = 4x^7 + 5x^6 - x^3 + 7x^2 - 3$




① compare exp. of to expo. of (q)

$8 \neq 7$ (Not equal) [if not equal than bigger poly expo. is added in final]

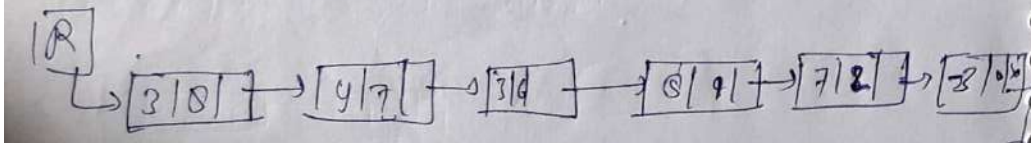
② move next node -

Compare exp. (p) to exp(q)

$6 = 6 \Rightarrow$ both are equal

③ $4 > 3$ (bigger poly exp. is added in final L.L]

Resultant




$\Rightarrow 3x^8 + 4x^7 + 3x^6 + 8x^9 + 7x^2 - 3$

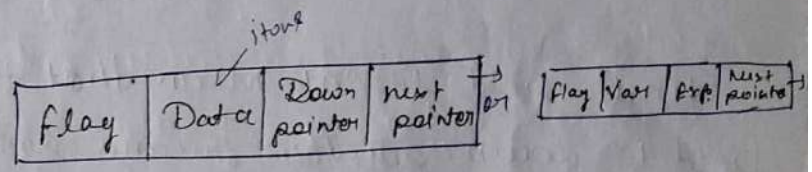
## Generalized link list ⇒

A generalized link list A is defined as of finite sequence of an greater than equal to zero elements. ($n >= 0$) $a_1, a_2, a_3 \ldots a_n$ such that $a_i$ are either atoms or list of atoms. Thus

$$A = \{a_1, a_2, a_3 \ldots a_n\}$$

where $n$ = total no. of nd

Now to represent such a list of atoms we will have certain assumptions about the node structure




where

flag = 1 means lower pointer exist.

= 0 means next pointer exist.

Down pointer ⇒ Down pointer is the address of node which is down of the current code.

Next pointer ⇒ Next pointer is the address of the node which is attach d at the next node.

(structure of Generalized l.L. ⇒

```
typedef struct
{char ci;   /* Data */
int ind;    /* flag */
```

Scanned by CamScanner

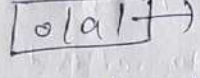
struct node *next, *down
{
gll;

/* next & down pointer */

Ex of gen L.L.

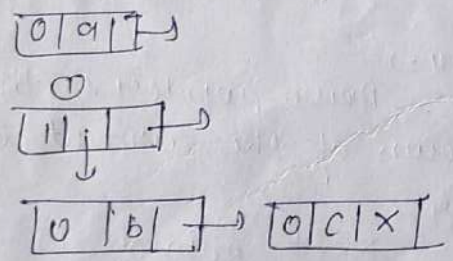
[-a, (b, c), d]

first node -

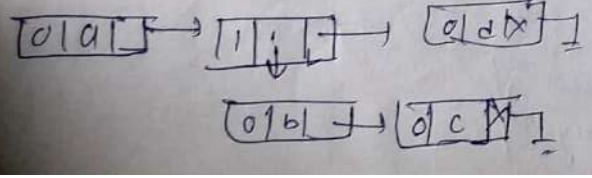


In this case the first field is zero the indicates

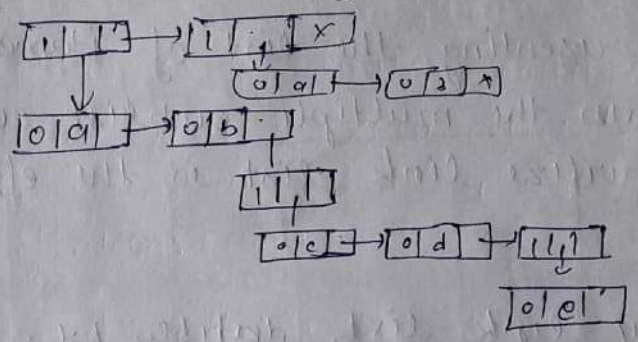
First field is one mean that the second field is down pointer. (means some list is starting)



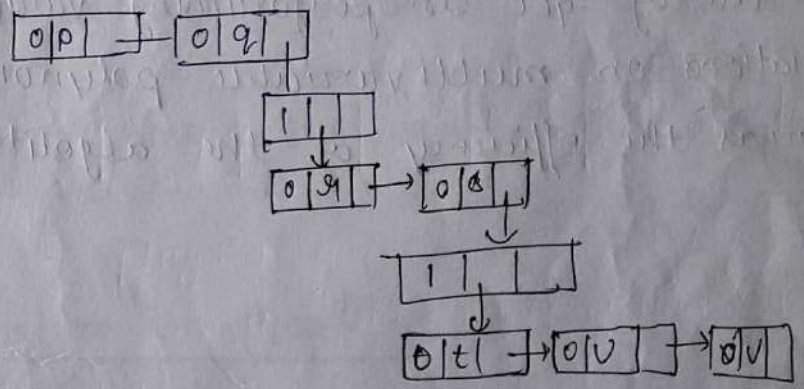
The above fig. indicates (A, B, C) In this way the gen L.L can be build.



(1) [a, b (c, d(e)), (a, b)]



(2) G = (P, q (r, s (t, u, v) w), x, y)



Scanned by CamScanner

## Advantages of generlize Link list:

① For representing the list of items which may contain the multiple sublist even the use of generlize link list is the efficient of

② As singly link list, doubly L.L. or the arrays are not efficient ways for multiple variable polynomials.

The use of GLL in performing various operations on multivariable polynomial increas the efficiency of the algorithm

# UNIT = 2
# [STACK]

A stack is an order list in which all insertions & deletions are made at one end Call the top of stack.

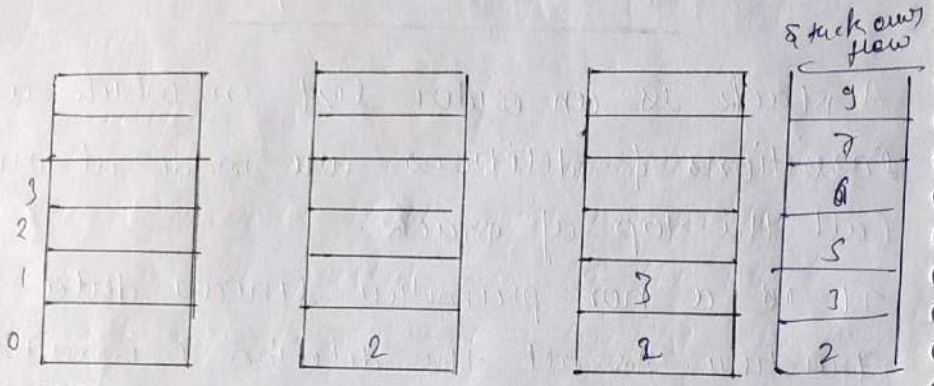
It is a non primitive linear data structure. As all the deletion & insertion in a stack is done from top of the stack then the last added element will be the first to be remove from the stack. This is the reason ~~vice~~ ~~cho~~ why stack is also called LIFO.

## Operation on Stack:

① **Push:** It is the process of adding new element to the top of stack is called push operation. pushing ~~the~~ element ~~in~~ the stack invoke adding of element, as the new element will be inserted at the top after every push operation the top is incremented by 1. (TOP = TOP+1)

Scanned by CamScanner

In case of stack is if already full & new element acominated, it is call stack full condition or stack overflow condition.




Top = -1 At the begin Stack is empty

Top = Top+1 insert 1st elem.

After inserting IInd elem. Top = Top+1

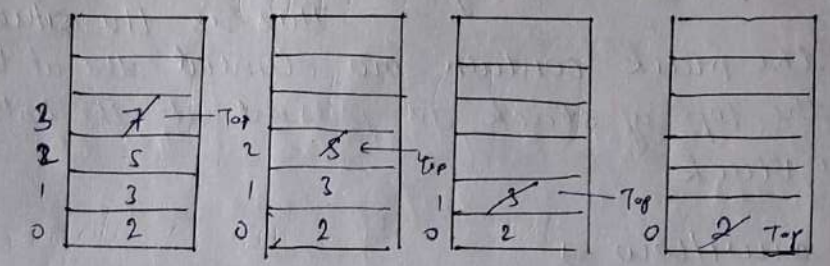
Top = Top+1

## POP

It is the process of deleting an element from the top of the stack is called pop operation.

After every pop operation the stack is decremented by 2.

If there is no element on the stack and deletion if perform then this will result into stack underflow condition or stack empty condition.




At top = 3 At the begin

Top = Top-1 deleting Ist elem

After deleting IInd elem Top = Top-1

Top = Top-1

## Stack terminology ⇒

① Context ⇒ The ~~toki~~ environment in which a function include argument values, local variables, global variables.

② Stack frame ⇒ The data structure containing all the data (arguments) return address needed these time a processor or function is called.

③ Max size ⇒

④ Top ⇒ The top of stack is use to check stack overflow or underflow condition. Initially top stores (-1). after every insertion in the stack the value of top is incremented by (1). And After every deletion the value of top is decremented by 1.

Scanned by CamScanner

deleted

Stack empty or underflow :-

This is the situation when the stack contain no element ~~the~~ at this time the top of stack is present at the bottom of the stack.

Stack overflow :-

This is the situation where the stack becomes full and no more elements can be pushed on to the stack at this point top is present highest location of the stack.

Algo for push operation:

1. Initilize Top = -1
2. Repeated steps 3 to 5.
3. Read items / elements
4. Top = Top + 1
5. Set [Top] = item
6. Print Stack "overflow".

Algo for pop operation :-

1. Repeat steps untill Top ≥ 0
2. Set Item = Stack [Top]
3. Top = Top - 1
4. If no item Deleted.
5. Print stack "underflow".

Scanned by CamScanner

Application of stack ⇒

① Special Expression Conversion.
② Expression evolution.
③ Decimal to Binary conversion.
④ Storing function calls.
⑤ Reversing a string.

Expression conversion ⇒

Data structure there are basically three types of expression

(i) Infix expression conversion.
(ii) Prefix " " .
(iii) Pofix " " .

Mostly ~~written~~ we done infix to postfix conversion which is much more easy to work and evaluate the expression. And follow mathematical rule for evaluation that is BODMAS. There are many types of expression conversion.

(i) Infix to postfix.
(ii) " " prefix.
(iii) Prefix to Infix.
(iv) Prefix to postfix.
(v) Postfix to Infix.
(vi) Postfix to prefix.

Infix expression conversion ⇒ This type of expression the operator is in the middle of

Scanned by CamScanner

their operants. The arrangment of operator & operants is as follows.

$\boxed{\text{Infix expression} = \text{operand 1 operator operand 2}}$

Ex ⇒ (a+b). (a+b) * (c-d). (a+b/e) * (d+f)

prefix expression conversion ⇒ In this expression the arrangement of operand & operator is as follows -

$\boxed{\text{prefix expression} = \text{operator . operand 1 operand 2}}$

Ex ⇒ (+ ab). (* +ab-cd)

postfix ⇒ In this expression the arrangment of operand & the operators is as follows -

$\boxed{\text{post fix expression} = \text{operand 1 operand 2 operator}}$

Ex ⇒ ~~(ab+), (ab+cd-*)~~

A+ [ (B+C) + (D+E) * F ) / G . give postfix form.

A + [ (BC+) + (DE+) * F ] / G
T S

when T = BC+ A + [T + (S * F)] / G

S = DE + A + [T + (SF *)] / G

Q = SF* . A + [T + Q] / G

P = TQ+ A + [ TQ+ ] / G

N = PG/ A + (P/G)

A + (PG/]

A + N

AN+

Highest priority - ( , *
/ , *
+ , -

Expanding the expression AN+

APG/+

ATQ+G/+

ABC+Q+G/+

ABC+SF*+G/+

ABC+DE+F*+G/+

① (A+B) * C/D + E ∧ F/G } postfix

② A + (B * C - (D/E ∧ F) * G) * H

③ A/(B ∧ C) + D . in prefix form

A/(∧BC) + D

T = ∧BC A/T + D

/AT + D

S = /AT

S + D

+SD

Expanding the expr^n -

+/AT D

+/A+∧BC

BODMAS → multiplication
↳ division → addition
Bracket of → subtr

(iii) $(A*B + (C/D)) - F$
(iv) $A/(B \wedge C) - D$ } prifix

(iv) $A/(B \wedge C) - D$

$A/(\wedge BC) - D$

$T = \wedge BC$ $\quad A/T - D$

$/AT - D$

$S = /AT$

$S - D$

$-SD$

$-/ATD$

$-/A\wedge BCD$

(iii) $A*B + (C/D) - F$

$= A*B + (/CD) - F$

$T = /CD$ $\quad A*B + T - F$

$*AB + T - F$

$Q = *AB$ $\quad Q + T - F$

$+QT - F$

$R = +QT$

$R - F$

Expanding $-$ $-RF$

$-+QTF$

Scanned by CamScanner

Infix to postfix

$\wedge \Rightarrow 3$
$*, / \Rightarrow 2$
$+, - \Rightarrow 1$

① Read the infix expression from left to right one charcter at a time.

② Initially push $ on to the stack for $ in stack said priority is -1.

③ If I/P symble is read ( then push it on to the stack).

⑧ If the I/P symbol is read is an operant is placed in a postfix operation.

④ If the I/P symble is read if operator then (a) Check if priority of operator is in stack is greater than the priority of incoming charater then pop that operator from the stack and place in the postfix operation.

(b) Otherwise push the operator being read, onto the stack.

(c) If we read input operator as ")" then of all the operators untill we get "(" & appened pop operators to postfix operation.

⑤ ~~pri~~ Finally pop the remaing contains in the stack. untill stack will become empty, append them to postfix operation.

⑥ ~~Bring~~ print the postfix operation as result

Scanned by CamScanner

| operator | In stack Priority | Incoming priority |
|---|---|---|
| + or - | 2 | 1 |
| * or / | 4 | 3 |
| ^ | 5 | 6 |
| ( | 0 | 9 |
| operand | 8 | 7 |
| ) | NA | 0 |

The priority of different operators when they are in stack will be called as in stack priority. And the priorities of different operators when they are from I/P will be called as incoming priorities.

Ex. A*B+C $ obtained postfix expression.

| Input | Stack | O/P |
|---|---|---|
| None | Empty | None |
| A | " | A |
| * | * | A |
| B | * | AB |
| + | pop * | AB* |
| C | | AB*C |

Algo. for Infix to prefix :-

① Reverse the I/P string.
② Examine the next element in the I/P
③ If it is operant, add it to the O/P string.
④ If it is closing parantheses push it on the stack.
⑤ If it is an operator then -
(a) If stack is empty push operation on the stack.
(b) If the top of stack is closing parantheses push operator on stack.
(c) If it has same or higher priority then the top of stack, push operator on stack.
(d) If else pop operator from the stack & add it to o/p string
⑥ If it is an open parantheses, pop operator from stack and add them to S until it closing parantheses is encountered.
⑦ pop and discard the closing parantheses
⑧ If there is more input goto step II
⑨ If there is no more I/P, unstack

Scanned by CamScanner

remaing operators and add them.

⑤ Reverse the O/P string.

## prefix to Infix transformation ⇒

① Reverse the prefix expⁿ.

② Read the reverse prefix expⁿ from left to right one character at a time.

③ If we read an operant then we push it on to the stak.

④ If we read operator then pop the first operation. Concatinate it, "c", all it a string1 then pop the second operant and contatinate it with string1 and call it as str2. And then concatinate str2 with closing parantheis. Thus an Infix expression operant1 to operant2 (op1 operator op2) gets from push this expⁿ on to the stack.

⑤ Go to step first untill the complete I/P is read.

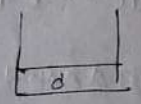
⑥ ~~Complex~~

① Convert prefix exp. * +ab - cd into ~~postfix~~ Infix evalute

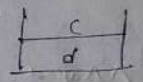
① Reverse the prefix -

dc - ba + *

② dc - ba + * Read operand & push it on stack




③ dc - bc + * Read operand & push it in stack

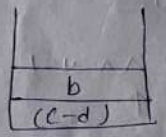



④ dc - bc + * Read operator & pop two operands & from infix exp

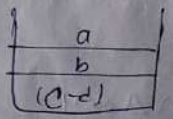
op1 = c
op2 = d

(c-d)

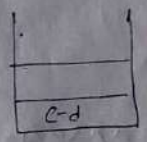
⑤ dc - ba + * Read operand & push it on stack




⑥ dc - ba + * Read operand & push it

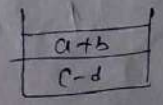



⑦ dc - ba + * Read operator & pop two operands & from infix expⁿ



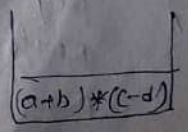

op1 = a
op2 = b
a+b

⑧ dc - bc + * Read operator & pop two operands & from




op1 = a+b
op2 = c-d

(a+b) * c-d




Scanned by CamScanner

Q. Convert abcde^^*+ into Infix & prefix using stack.

| Input | ~~stack~~ Action | Stack |
|---|---|---|
| abcde^^*+ ↑ | push | a |
| abcde^^*+ ↑ | push | ab |
| abcde^^*+ ↑ | push | abc |
| abcde^^*+ ↑ | push | abcd |
| abcde^^*+ ↑ | push | abcde |
| abcde^^*+ ↑ | pop two operands perform Infix operation push the results | abc(d^e) |
| abcde^^*+ ↑ | pop two operands perform Infix operation push the results | ab(c^(d^e)) |
| abcde^^*+ ↑ | pop two operands perform Infix operation push the results | ~~abcd~~. a(b*(c^(d^e))) |
| abcde^^*+ ↑ | pop two operands perform Infix operation. push the results | (a+(b*c^(d^e))) |

pop & print the result.

In prefix form of abcde^^*+

| Input | Action | stack |
|---|---|---|
| abcde^^*+ ↑ | push | a |
| abcde^^*+ ↑ | push | ab |
| abcde^^*+ ↑ | push | abc |
| abcde^^*+ ↑ | push | abcd |
| abcde^^*+ ↑ | push | abcde |
| abcde^^*+ ↑ | pop two operands perform prefix operation push the results | ab^c(^de) |
| abcde^^*+ ↑ | pop two operands perform prefix perform push the results | a^b(^c(^de)) |
| abcde^^*+ ↑ | pop two operands perform prefix perform push the results | a*b(^c(^de)) |
| abcde^^*+ ↑ | pop two operands perform prefix perform push the results | +a(*b(^c(^de))) |

Scanned by CamScanner

consider following postfix notation

Que Convert 752+*415-/- into P  CT-2

① find the value of expression.

② find the equivalent prefix notation of above expression.

| Input | Action | Stack |
|---|---|---|
| 752+*415-/- ↑ | Push | 7 |
| 752+*415-/- ↑ | Read 5, Push | 75 |
| 752+*415-/- ↑ | Read 2, Push | 752 |
| 752+*415-/- ↑ | pop two operands (5+2=7) perform Infix operation push the result | 77 (op1 op2) |
| 752+*415-/- ↑ | pop two operands perfor Infix (7*7=49) push the result | 49 |
| 752+*415-/- ↑ | read 4, push | 494 |
| 752+*415-/- ↑ | read 1, push | 4941 |
| 752+*415-/- ↑ | read 5, push | 49415 |
| 752+*415-/- ↑ | pop two operands (1-5=-4) perform Infix form push result | 4940-4 |
| 752+*415-/- ↑ | pop two operands perfor Infix operation (4/-4 = -1) push result | 49 -1 (op1 op2) |
| 752+*415-/- ↑ | pop two operands perform infix operation (49-(-1)=50) push result | 50 |

Into prefix perfor

| Input | Action | stack |
|---|---|---|
| 752+*415-/- ↑ | push | 7 |
| 752+*415-/- ↑ | push | 75 |
| 752+*415-/- ↑ | push | 752 |
| 752+*415-/- ↑ | pop two operands perform prefix form push the | 7(+52) |
| 752+*415-/- ↑ | | (*7(+52)) |

Scanned by CamScanner

| | | |
|---|---|---|
| 752+*415-/- | push | 4 / *7(+52) |
| 752+*415-/- | push | 1 / 4 / *7(+52) |
| 752+*415-/- | push | 5 / 1 / 4 / *7(+52) |
| 752+*415-/- | pop two operands<br>perform prefix operation<br>push result | -51 / 4 / *7(+52) |
| 752+*415-/- | pop two operands<br>perfor prefix operation<br>push result | 4(-51) / *7(+52) |
| 752+*415-/- | pop two operands<br>perform prefix operation<br>push result | -*7(+52)/4(-51) |

## prefix to postfix transformation :

**Algo**

① Reverse the given prefix expression.

② Read this ~~pre~~ reverse prefix exp^n from left to right one character at a time

③ If we read operand than push it on to the Stack.

④ If we read operator than pop two operands, first operand call it as $OP_1$ than second operand call it as $OP_2$ Concatinate $OP_1$ & $OP_2$ with operator so that postfix expression "$OP_1$ $OP_2$ operator" will get form.

⑤ Go to step I untill the complete ~~is~~ I/P is read.

Ex:- Prefix to postfix :- *ab+cd-

1 Reverse the exp^n :- dc-ba+*

| Input | Action | stack |
|---|---|---|
| dc-ba+* | push | d |
| dc-ba+* | push | c / d |
| dc-ba+* | pop two operands<br>perform postfix operation<br>push the result | cd- |
| dc-ba+* | push | b / cd- |
| dc-ba+* | push | a / b / cd- |
| dc-ba+* | pop two operands<br>perform postfix operation<br>push the result | abt+ / cd- |

Scanned by CamScanner

| | | |
|---|---|---|
| dc-ba+* ↑ | pop two operands<br>perfor postfix operation<br>push the result | ab+cd-* |

postfix to infix ⇒ $\boxed{OP_2 \text{ operator } OP_1}$

① Read the postfix exp$^n$ one character at a time.

② If we read operand then push it on to the stack.

③ If we read operator then pop first operand concatinate it "C" call it as str1. Then pop second operand and concatinate it with str1, call it as str2. And then concatinate str2 with closing brackets. Then infix exp$^n$ "$OP_1$ operator $OP_2$" gets form. push this exp$^n$ on to the stack.

④ Go to step 1 untill the complete I/P is read.

Ex ⇒ AB + CD - *

| Input | Action | Stack |
|---|---|---|
| AB+CD-* ↑ | push | A |
| AB+CD-* ↑ | push | B, A |
| AB+CD-* ↑ | pop two operands | A+B |
| AB+CD-* ↑ | push | C, A+B |
| AB+CD-* ↑ | push | D, C, A+B |
| AB+CD-* ↑ | pop two operands<br>perform infix operation<br>push the result | C-D, A+B |
| AB+CD-* ↑ | pop two operands<br>perform infix op<br>and push | A+B*C-D |

postfix to prefix ⇒ $\boxed{A \cdot * + AB - CD}$

| Input | Action | Stack |
|---|---|---|
| AB+CD-* ↑ | push | A |
| AB+CD-* ↑ | push | B, A ($OP_2$, $OP_1$) |
| AB+CD-* ↑ | pop two operands<br>perform prefix operation<br>and push the result | +AB |
| AB+CD-* ↑ | push | C, +AB |
| AB+CD-* ↑ | push | D, C, +AB |

Scanned by CamScanner

AB+CD-*

↑

pop two operands

perform prefix operation

push result

| |
|---|
| -CD |
| +AB |

AB+CD-*

↑

pop two operands

perform prefix operation

push result

| |
|---|
| *+AB-CD |

Scanned by CamScanner

Evaluation of Expression :-

As we have seen how ~~given~~ to convert given infix exp^n to postfix form. It's time to learn & evalution of postfix exp^n.

Again used of stack is necessary postfix exp^n evaluation

Here we will use the stack for storing operands rather than operators. operands are taken as numeric values.

Algo for evalution of postfix :-

① Read the postfix exp^n from left to right.

② If the inputs symbol is an operand then push it on the stack.

③ If the operator is read pop two operands and perform arithmatic operations, If operator is: + then result = $Op_1 + Op_2$

- " " = $Op_1 - Op_2$

* " " = $Op_1 * Op_2$

/ " " = $Op_1 / Op_2$

④ Push the result on to the stack.

⑤ Repeat steps 1 to 4 till the postfix is not over.

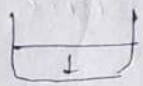
Scanned by CamScanner

Ex ⇒ AB+C - BA+C $-

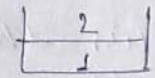
if given is A=1, B=2, C=3

AB+C - BA+C $-

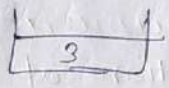
① AB+C - BA+C $-




② AB+C - BA+C $-

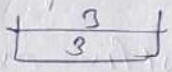



③ AB+C - BA+C $-

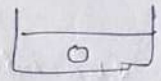



⇒ 1+2 = 3

④ AB+C - BA+C $-

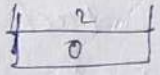



⑤ AB+C - BA+C $-

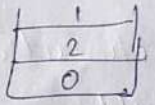



⇒ 3-3=0

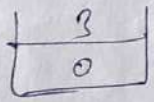
⑥ AB+C - BA+C $-




⑦ AB+C - BA+C $-

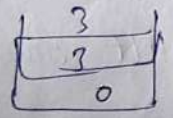



⑧ AB+C - BA+C $-

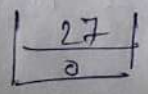



⇒ 1+2 = 3

⑨ AB+C - BA+C $-



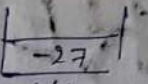

⑩ AB+C - BA+C $ -




$ = $3^3 = a^b$
= 27

AB+CD-*

⑪ AB+C - BA+C $ -




0-27 = -27

perform evaluation of postfix expⁿ using stack!

Ex ABC + * DE/-

where A = 5
B = 6
C = 2
D = 12
E = 4

| | | |
|---|---|---|
| ① ABC + * DE/- | push | A ⇒ 5 |
| ② ABC + * DE/- | push | AB ⇒ 5,6 |
| ③ ABC + * DE/- | push | ABC ⇒ 5,6,2 |
| ④ ABC + * DE/- | pop two operands | ~~ABC*~~ 5,8 |
| ⑤ ABC + * DE/- | pop two operands | ~~ABC+*~~ 40 |
| ⑥ ABC + * DE/- | push | 40, 12 |
| ⑦ ABC + * DE/- | push | 40, 12, 4 |
| ⑧ AB + * DE/- | pop two operands | 40, 3 ⇒ 12/4 = 3 |
| ⑨ AB + * DE/- | pop two operands | 37 |

⑨ convert prefix to postfix *+AB-CD

Scanned by CamScanner

# RECURSION

**Recursion →** It is a programing technic in which function call itself using some input value.

**Types of recursion →**

Recursion of two type depending on whether a function call itself from within itself is called Direction recursion

```
int abc ( )
{
  ----
  ----
  abc ( );
}
```

**Indirect ⇒**

When two function call one another mutually it is called indirect recursion.

```
int abc ( )
{
  ----
  ----
  xyz ( );
}
int xyz ( )
{
  abc ( )
  ----
}
```

Recursion may be catagorized as follows-

1. Linear recursion.
2. Binary recursion.
3. Multiple recursion.

**Linear recursion ⇒** Linear recursion ~~within~~ begins by testing for a set of base cases there should be atleast one.

(i) Perform a single recursive call.

(ii) This recursive state may involve a test that decide which of several possible recursive call to make.

(iii) Each define possible recursive call so it makes progress towards a base case

**Algo →** Input ⇒ An int array A and an integer n=1 such that A has atleast ~~one~~ n elements.

output ⇒ If n=1 then
return A[0]
else return linear sum (A, n-1 + A[n-1])

int

Binary recursion ⇒ Binary recursion occur whenever there are two recursive call for each non base case.

multiple recursion ⇒
we make not just one or two many recursive calls. On

Ex ⇒ cat + Dog = Mat
mat + Let = Zat
dog + mat = cat

Advantage =
More efficient in terms of speed & execution time.

Disadvantage ⇒
It consume more storage space because the recursive call along with automatic barriers are stored on the stack.

Difference between Itration and recursion

| Itration | Recursion |
|---|---|
| ① Itration is a process of executing certain set of instruction repeatedly without calling the function. | It is a process of executing certain set of instruction repeatedly by calling the self function continuously. |
| 2. The itrative function are implemented with help of for, while, do while programing loops collustructs | ② Instead of making use of for, while, do while the repeation in code execution obtained by calling the same function again & again. |
| ③ The itrative methods are more efficient because of better execution speed. | The recursive methods are less efficient. |
| ④ Memory utilization by itration each less | ④ memory utilization is more in recursive functions. |
| ⑤ It is simple to implement | It is complex to implement. |
| ⑥ The line of code is ~~more~~ more than we used itration. | ⑥ Recursive method bring compactness in the program. |

Scanned by CamScanner

# Tower of Hanoi Problem (TOH)

It is a gaming problem. There will be three pegs & and three different size disc.

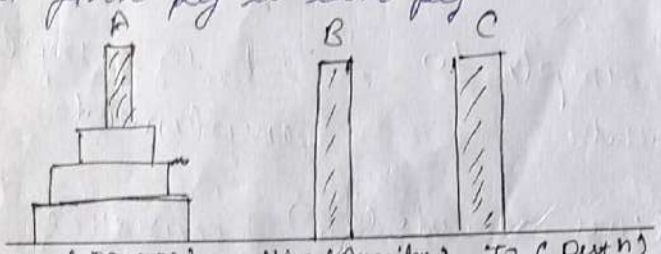
Three disc are placed on peg A. The arrangement of the disc is such that every smaller disc is placed on the larger disc.

The conditions are -

(i) Only the top disc on any peg may be move to any other peg.

(ii) A larger disc should never rest on the smaller one

(iii) There are three pegs and n no. disc placed on over the other in decreasing order. The objective of this game is to move disc one by one from first peg to last peg




From (source) Using (Auxiliary) To (Destn)

⇒ To solve this prob. we will follow three steps:

we will use general notation T(N, Beg, AUX, End)

where T = denotes procedure

N = no. of disc

Beg = Source / Initial peg  End = Desth / Final peg

AUX = Auxialary peg

Steps: ⇒

T(N-1, Beg, End, AUX) → move Top N-1 disc from begin to AUX peg

T(1, Beg, AUX, END) → move one disc from begin to end peg

T(N-1, AUX, Beg END) ⇒ move N-1 disc from AUX to N

T(3, A, B, C)

S1 = T(2, A, C, B)

S+2  T(1, A, B, C)

S+3  (T2, B, A, C)

T.8 (3, A, B, C) → T(2, A, C, B); T(1, A, B, C) → A→C; T(2, B, A, C)

T(2, A, C, B) → T(1, A, B, C) → A→C; T(1, A, C, B) → A→B; T(1, C, A, B) → C→B

T(2, B, A, C) → T(1, B, C, A) → B→A; T(1, B, A, C) → B→C; T(1, A, B, C) → A→C

Algo T(N, Beg, AUX, END)

Begin

{ if N=1 then

print (Beg → END);

else

Call T(N-1, Beg, End, AUX)

Call T(1, Beg, AUX, End)

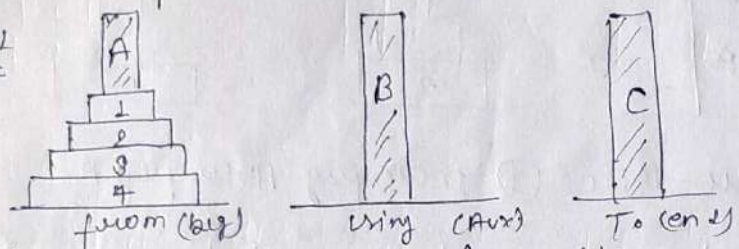
Call T(N, AUX, Beg, END)

Scanned by CamScanner

① ~~write a algorithm~~

Find the solution of four disc to the tower of Hanoi problem.

Step-1

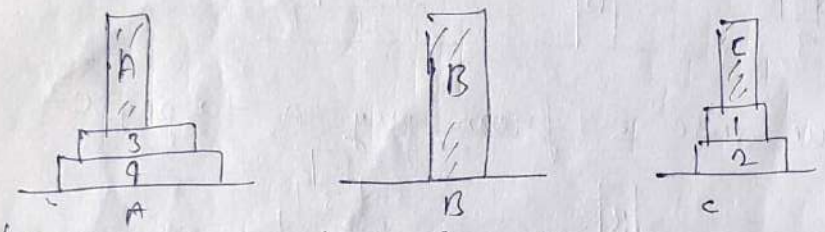



Step-2 If there are four disc.

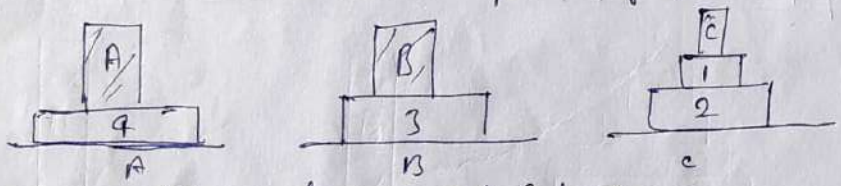
Move disc 1 from A to B

move disc 2 from peg A to C

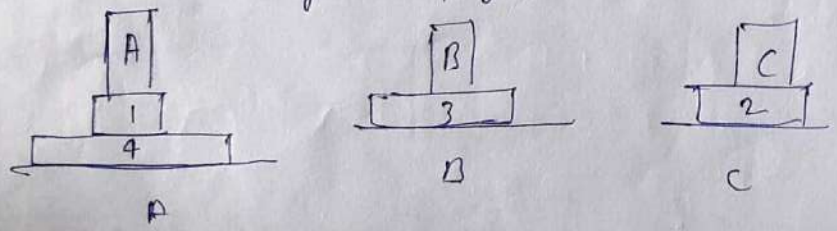
Move disc 1 forom peg B to C



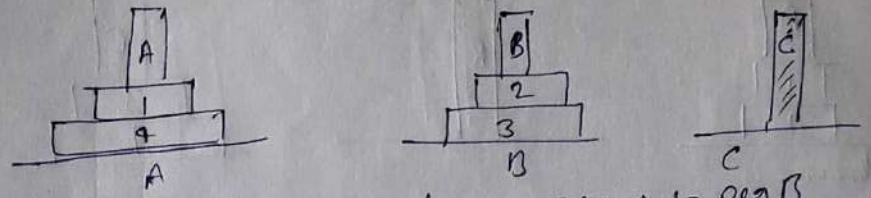

Step-3 Move disc 3 from peg A to B



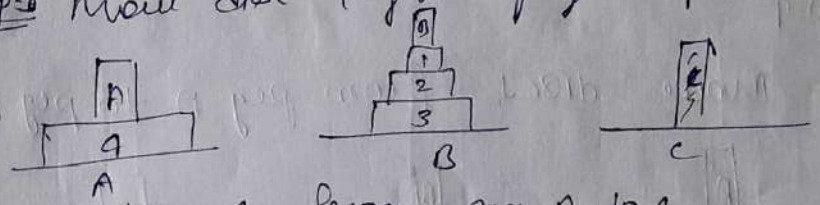

Step-4 move disc 1 from peg C to A



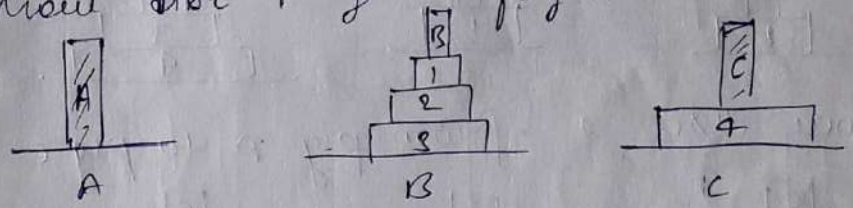

Step-5 Move disc 2 from peg C to B



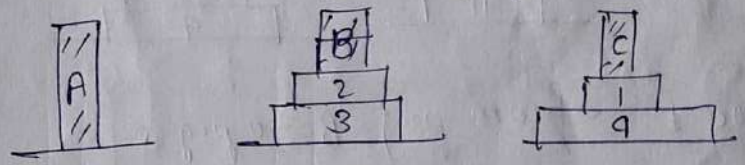

Step-3 Move disc 1 from peg A to peg B



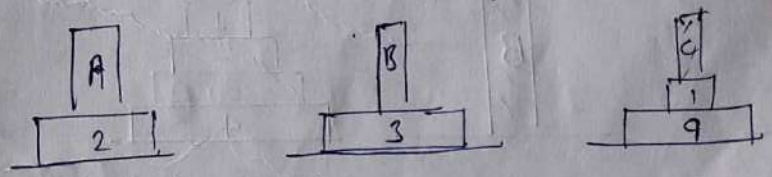

move disc 4 from peg A to C



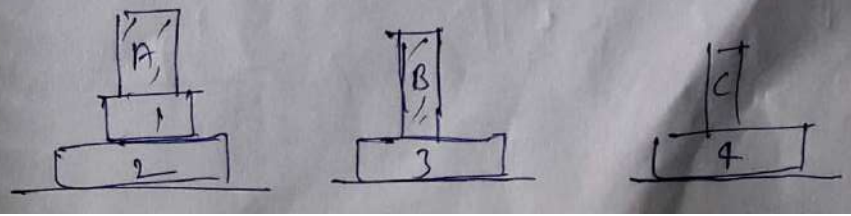

Move disc 1 from peg B to C




Move disc 2 from peg B to peg A

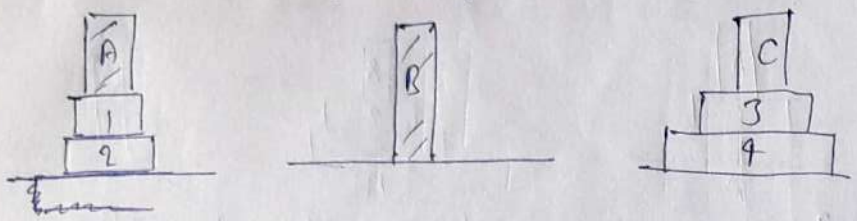



Move disc 1 from peg C to A



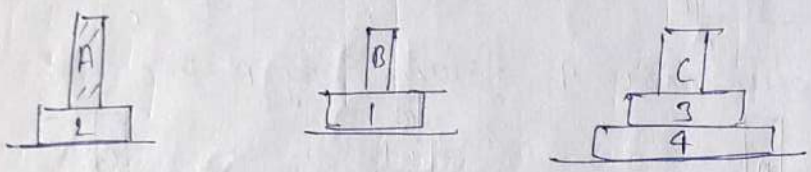

Scanned by CamScanner

Move disc 3 from peg B to peg C

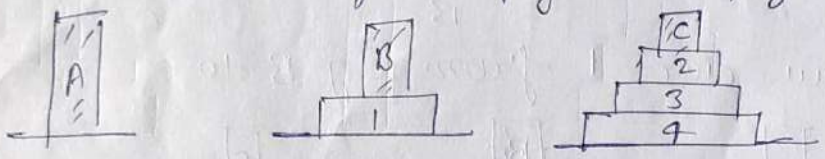



Step 4

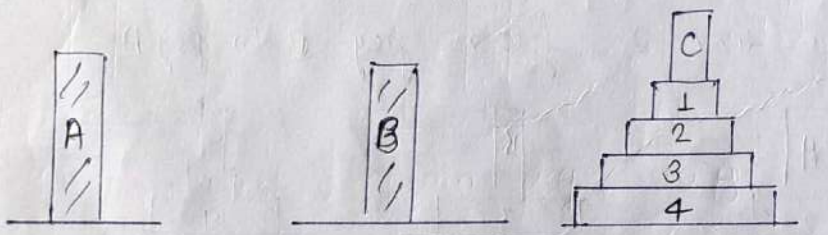
Move disc 1 from peg A to peg B




Move disc 2 from peg A to peg C



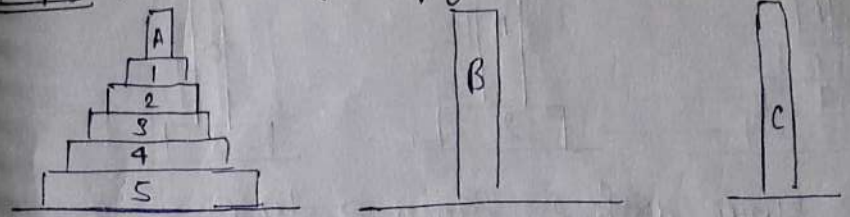

Move disc 1 from peg B to peg C



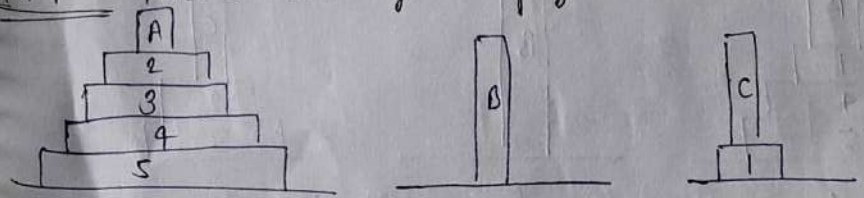

Answer

for 5 disc

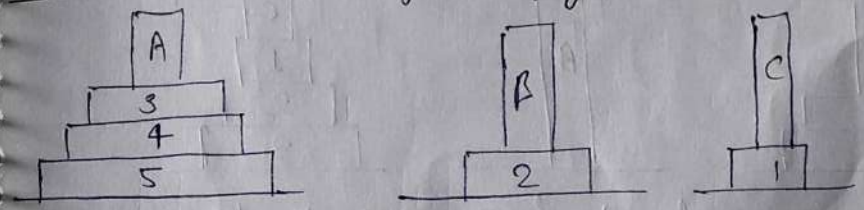
Step-1 Move disk ~~from peg A~~



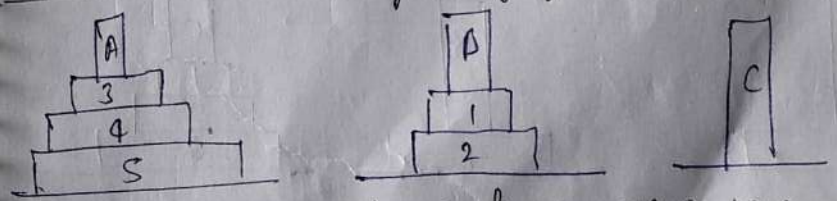

Step-1 Move disk1 from peg A to C



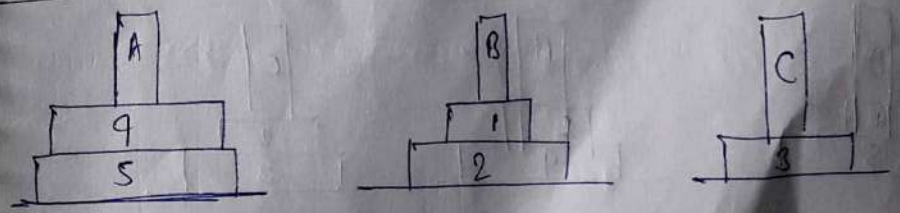

Step-2 Move disk2 from peg A to B




Step-3 Move disk1 from peg C to B

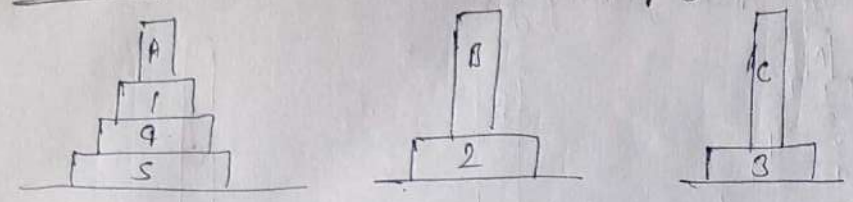



Step-4 Move disk 3 from peg A to C

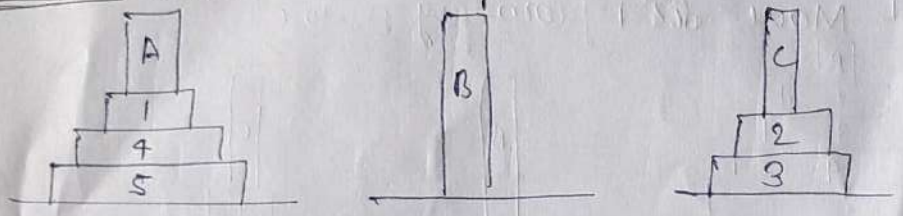



Scanned by CamScanner

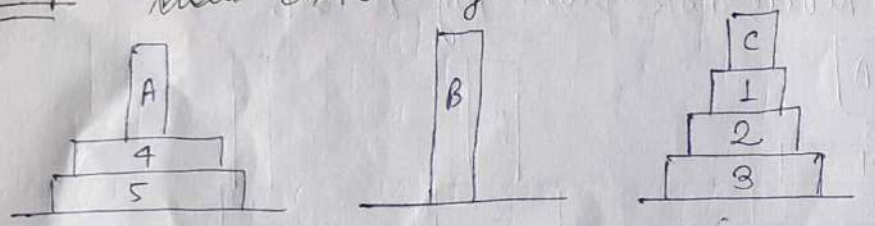
Step-5 Move disk 1 from peg B to A



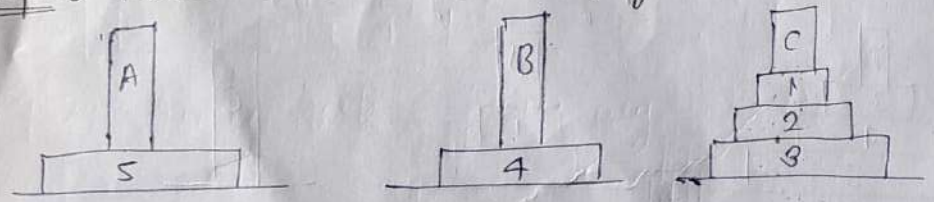

Step-6 Move disk 2 from B to C



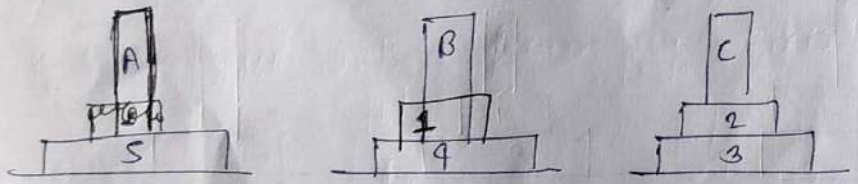

Step-7 Move disk 1 from A to C



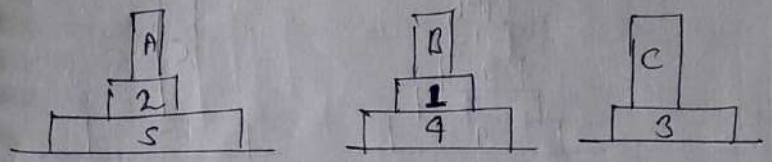

Step-8 Move disk 4 from peg A to C



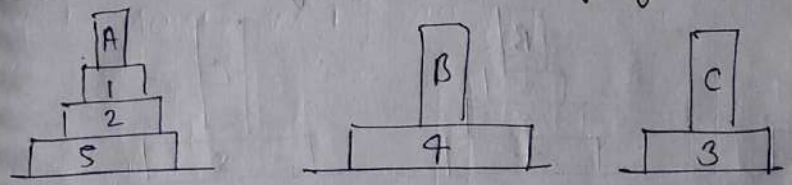

Step-9 Move disk 1 from peg C to B



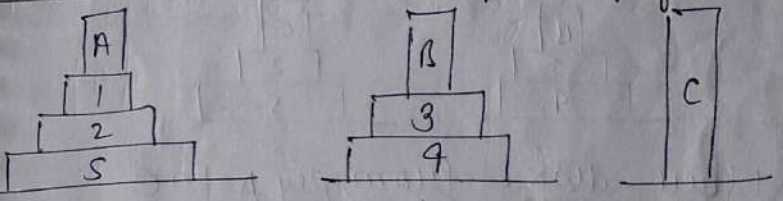

Step-10 Move disk 2 from peg C to A



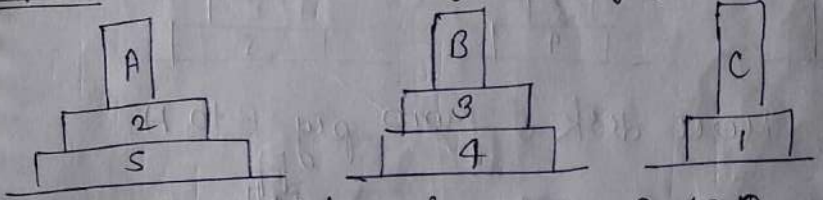

Step-11 Move disk 1 from peg B to C



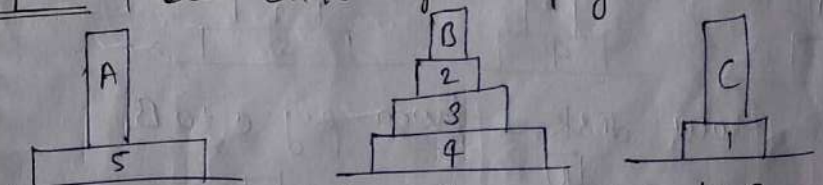

Step-12 Move disk 3 from peg C to B



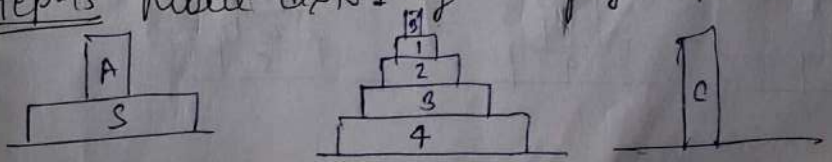

Step-13 Move disk 1 from peg A to C




Step-14 Move disk 2 from peg A to B

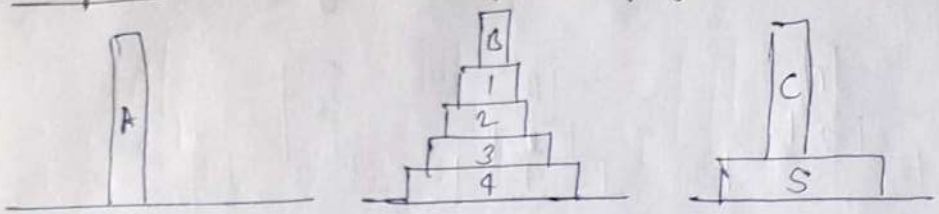



Step-15 Move disk 1 from peg C to B

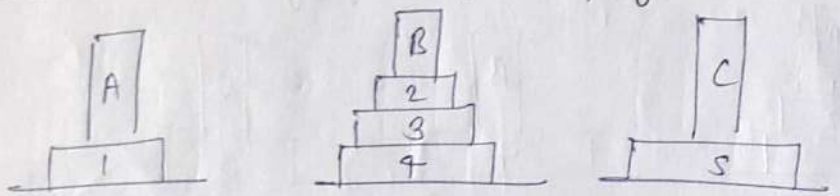



Scanned by CamScanner

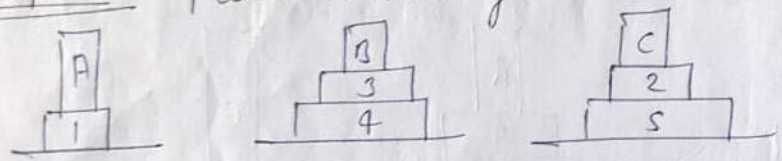
**Step-16** Move disks from peg A to C



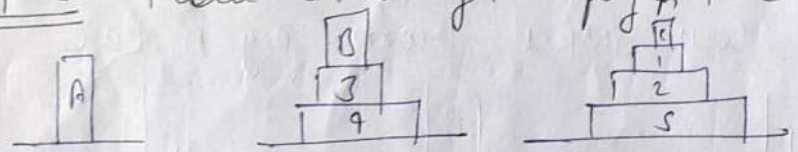

**Step-17** Move disk 1 from peg B to A



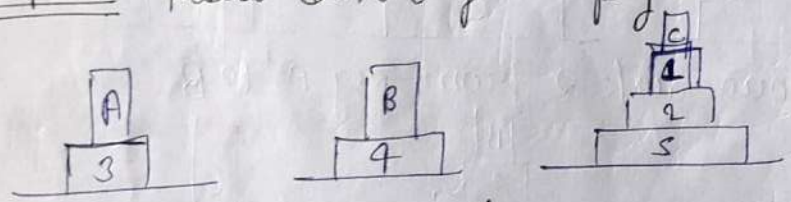

**Step-18** Move disk 2 from B to C



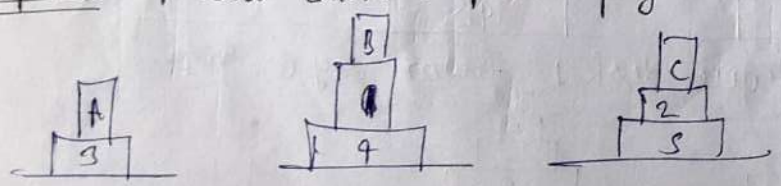

**Step-19** Move disk 1 from peg A to C




**Step-20** Move disk 3 from peg B to A




**Step-21** Move disk 1 from peg C to B




No. of steps required for 'n' disc $= 2^n - 1$

If there are 'n' no. of disk, by this we get a eqn.

$$T(n) = 2\{T(n-1)\}+1$$
$$T(n) = 2\{2T(n-2)+1\}+1$$
$$T(n) = 2^2[T(n-2)+3$$
$$T(n) = 2^3[T(n-3)]+7$$

If there are 'K' no. of disks then.

$$\boxed{T(K) = 2^K + (n-K) + 2^K - 1}$$ → Generalised eq

Hence total no. of steps for 'N' disks -

$$T(K) = 2^K . T(n-K) + 2^K - 1$$

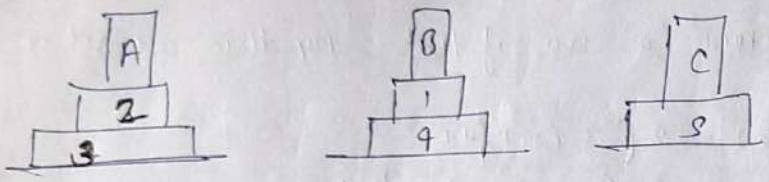
where $K = n$

$$T(n) = 2^n T(n-n) + 2^n - 1$$
$$= 2^n . T \times 0 + 2^n - 1$$
$$= 2^n - 1$$

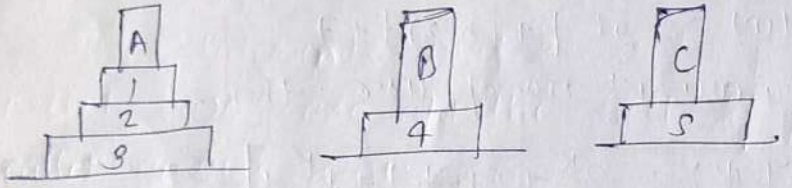
$$\boxed{T(n) = 2^n - 1}$$

Scanned by CamScanner

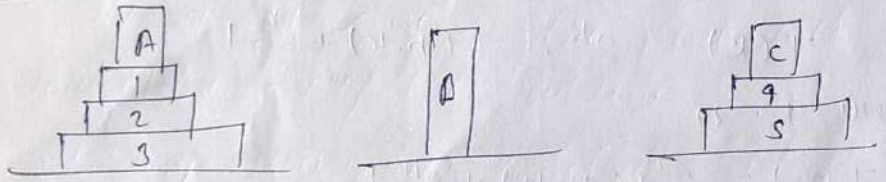
Step-22 Move disk 2 from peg C to A



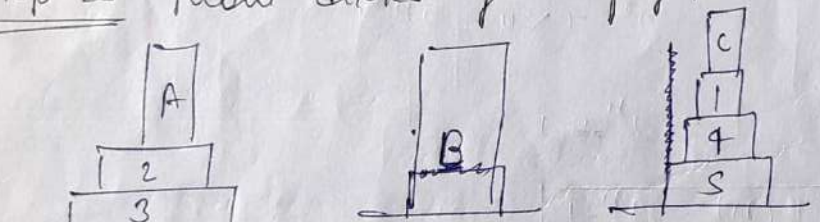

Step-23 Move disk 1 from peg B to A



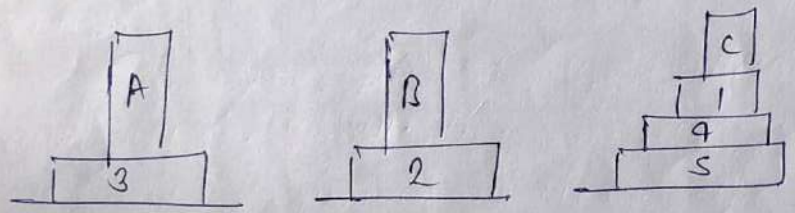

Step-24 Move disk 4 from peg B to C



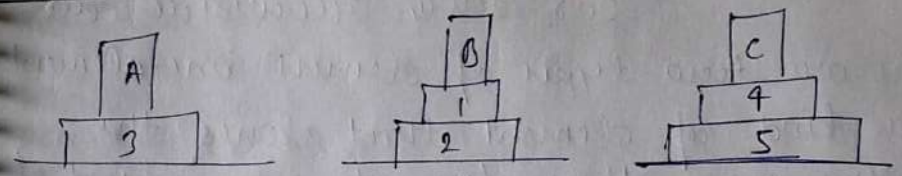

Step-25 Move disk 1 from peg A to C



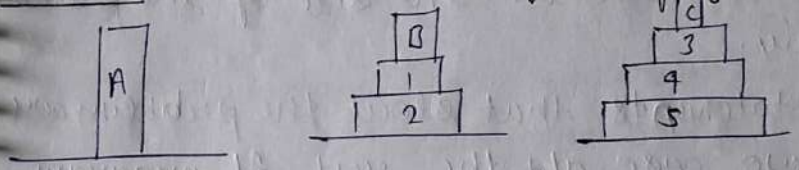

Step-26 Move disk 2 from peg A to B



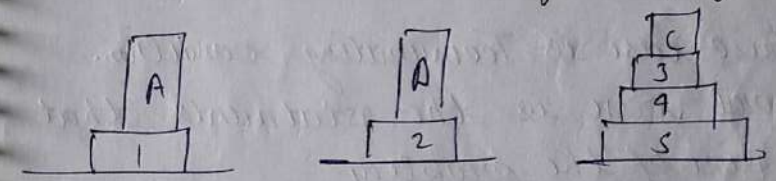

Step-27 Move disk 1 from peg C to B



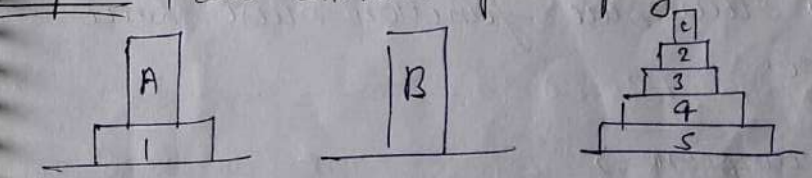

Step-28 Move disk 3 from peg A to C



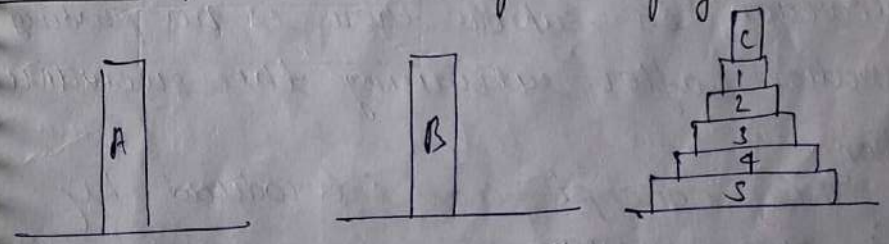

Step-29 Move disk 1 from peg B to A




Step-30 Move disk 2 from peg B to C




Step-31 Move disk 1 from peg A to C




Answer

Scanned by CamScanner

principle of recursion ⇒

⇒ For every recursive call there are two types of element one element is a kind of element that slows the problem and other element is a kind of element that reduces the size of the problem recursively.

⇒ The statements that slove the problem are called base case and the rest of program called general case.

⇒ The base case is Terminating condition.

⇒ The general case is the statments that reduce the size of the problem.

⇒ Every recursive function must have base case.

## Tail Recursion ⇒

Tail recursion is a kind of recursion in which there is no pending of operation after returning the recursive function.

This concept was introduced by David. H. D. Warren. This is a special kind of recursive approach which helps in improving the space & time efficiency of recursive function.

program using tail recursion ⇒

```
# include <stdio.h>
# include <conio.h>
void main ()
{
int fact (int n);
int n
Printf ("program finding factorial using tail
                recursion");
printf ("\n Enter the no. \n");
scanf ("%d", &n);
printf ("The fact = %d", fact (n));
getch ();
}
int fact (int n)
{
int my new - fact (int n, int f);
return my new fact (n, 1); → No pending opera.
}
int my new fact (int n, int f);
{ if (n==1)
return f;
my new fact (n-1, n*f);
}
```

Scanned by CamScanner

# Queue

The Queue is defined as ordered collection of non primitive linear data structure that has two end "front" end and "rear" end. From the front end one can delete the elements and from the rear and one can insert the element.

Queue works on FIFO.

Representation of Queue ⇒

(i) Array representation (Static implementation)
(ii) Link list representation (Dynamic implementation)

## Array representation ⇒

(i) In an array representation front will be an index variable which will hold the index of the element which will be deleted.

$\boxed{\text{Initial value of front} = -1}$

(ii) Here, rear is also an index variable which will hold the index of last element in the Queue

$\boxed{\text{Initial value of rear} = -1}$

$$\boxed{\text{Total no. of element present in a Queue} = \text{front} + \text{Rear} - 1}$$

* If rear < front then there will be in the Queue i.e Queue will always empty

## Link list representation ⇒

In this representation front will be a pointer variable which will hold the add. of the element on the node i.e. to be deleted.

$\boxed{\text{Initial value of front} = \text{NULL}}$

(ii) Here, Rear is the also a pointer variable which will hold the add. of last element in the node

$\boxed{\text{Initial value of Rear} = \text{NULL}}$

## Operations on a Queue →

* **Insertion Operation ⇒**

(i) In this operation the new element into added into the Queue from the rear end.

(ii) It follows the following steps
(a) checking the overflow
(b) If Queue is empty then front will be send to 0
(c) Increment the rear and put the item in rear position.

**Algo** procedure (Q, item, size, front, rear)

Q = Linear array or
item =
size = max. no. of element Queue can be held.
front, rear = index variable

Scanned by CamScanner

① If rear = size -1

(i) print it is over flow condition.

(ii) Other wise go to step second.

Step 2 If rear = -1

(i) front = front + 1 i.e front = 0

(ii) Rear = Rear + 1

(iii) Q[rear] = item

Step 3 Exit.

Deletion in Queue ⇒

In this operation we incremented the front and value. It follows the following steps.

(i) Check underflow.

(ii) Checking for single element.

(iii) Incrimenting front.

Algo

procedure ( Q, item, front, rear)

Step 1 If front = -1

(i) print underflow condition.

(ii) Otherwise go to Step 2.

St2 else if front = rear & front ≠ -1

(i) Print the deleted item Q[front]

(ii) front = -1 (iii) if rear = -1

(iv) Otherwise goto step 3.

Step 3 print the deleted item Q[front]

front = front + 1

Step 4 Exit.

Traversal OR Display ⇒

In traversal method visiting of elements starts from one end to till the rear end, by which we can say the elements are traverse from one location to another location. It follows FIFO principle

Algo procedure Q, front, rear)

Step 1 If front = -1

(i) printf "underflow" condition

(ii) otherwise goto step 2.

Step 2 for i = front to rear

printf Q[i]

end for

end if

Step 3 Exit.

Step 4 Update ⇒ updating means modifies the existing data in the Queue

Scanned by CamScanner

Algo procedure ( Q, item, front, rear, pos)
↓
(position at which item will be updated)

Step-1 pos = pos-1

Step-2 If (pos > front) & (pos <= rear)

print & Q [pos-1] = item;

otherwise goto step 3

Step3 print invalid condition

Link list repr

(i) It is possible using link list also.

(ii) The main advantage in link list representation is that we need not to have to worry about size of the Queue.

(iii) As in link list representation we can create as many nodes as we want so there will not be a Queue full condition at all.

(iv) It is similar to a link list but the difference b/w the two is in Queue, the left most node called front node & right node called rear node, and we can't remove any aribitary node from Queue.

(v) We have to remove front node always.

Structure

```
typedef struct node
{ int data;
  struct node next;
}Q;
```

Circular Queue ⇒

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

In other words.

If small n elements then after inserting an element at last location (i.e in the n-1 place) of the array then the next element will be inserted at very first location (i.e location which subscript 0) of the array.

It is possible to insert new element iff those locations (slots) are empty.

We can say that a circular Queue is one in which a first element comes just after the last element.

It can be viewed as rush OR loop of

circle in which the two ends of the queue are connected together.

Assumptions of circular Queue ⇒

(i) Front will always be pointing to the first element.

(ii) If front = rear the Queue will be empty.

(iii) Each time a new element is inserted into the Queue, rear is incremented by 1

$$\boxed{Rear = Rear + 1}$$

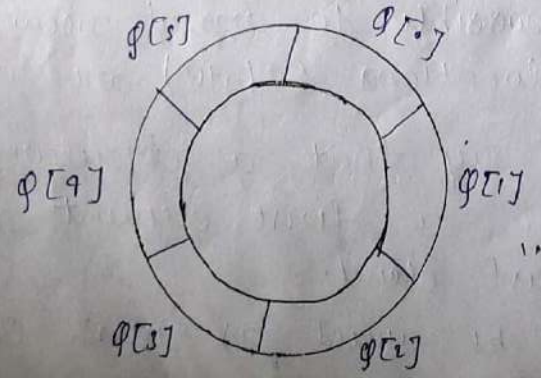
(iv) Each time an element is deleted from the Queue, the value of front is incremented by 1

$$\boxed{Front = front + 1}$$

(v) There is a formula which has to be applied for setting the front and rear pointers.

$$\boxed{Rear = (Rear + 1) \% Size}$$

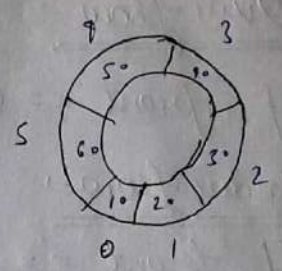
$$\boxed{Front = (Front + 1) \% Size}$$




"circular" "Queue"

$$Rear = (Rear + 1) \% Size$$
$$= (4+1) \% 6$$
$$= 0$$

$$front = (front + 1) \% Size$$
$$= (8+1) \% 9$$
$$= 0$$




Q suppose a Queue is maintained by circular Queue with n = 1 memory cells.

Find the no. of elements in the Queue when

(i) front = 9 & Rear = 8.

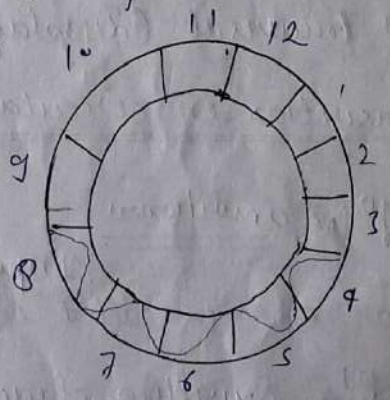
(ii) front = 10 & Rear = 3

(iii) Front = 5 & Rear = 6

(i) Total no. of element = 5

(ii) " " " " = 6

(iii) " " " " = 2




Advantages ⇒

(i) A circular Queue over comes the problem of utilized space it in linear Queue implement as array.

(ii) Like a linear Queue, circular Queue also having front & rear end for deleting and inserting an element.

Scanned by CamScanner

## Over flow ⇒

If front == (rear+1) % Size

## Under flow ⇒

If front == -1 then under flow occurs.

## Operation on circular Queue ⇒

(i) Insertion.
(ii) Deletion.
(iii) Traversal (Display).

## Insertion in circular Queue ⇒

Algo for Insertion ⇒ Inputing the element in the rear end is called insertion.

Step 1 Overflow checking.
Step 2 checking the first element for insertion i.e is it the first element that is going to insert.
St 3 Find the next rear position.
S-4 Input the item or element in the calculated rear position.

## Algo

procedure CQ (Q, front, rear, size, item)
front = index variable. initial value = -1
Rear = " " " " = -1
Size = Maxsize of CQ
item = elements to be inserted.

St. 1 If (front == (rear+1) % Size)
(i) print overflow.
(ii) Exit
[End if]

Step 2 if (rear == -1)
as front = 0
[End if]

Step 3 rear = (rear+1) % size

S-4 CQ [rear] = value / item

S-5 Exit

or

St 1 Else: Take the value
S.2 If (front == -1)
set front = rear = 0
else
S 3 rear = (rear+1) % size
Assign value CQ[rear] = item
[End if]
S 4 Exit

## Deletion in CQ ⇒

Deletion means deleting the element from the front end.

Step 1 Checking for underflow i.e (front == -1)
Step 2 Checking if there is exist only one element, if is then intilize front = rear = -1
St 3 Otherwise increment front value i.e front = (front+1) % size

Here, after going for next front value the element i.e to be deleted should be stored in a index variable.

Algo procedure C

S-1 If (front == -1)
(write underflow & exit)

S-2 else: item = Queue (front)
if (front == rear)
Set front = -1
rear = -1
else front = (front + 1) % size
[End if]

Step 3 Exit

Traversal in CQ ⇒

Visiting each element exactly once.

Algo Display (CQ, front, rear, i, size)
i = for loop control variable or counter variable.

S-1 if front = -1 then print underflow

S-2 if (rear ≥ front)
i: for (i = front ; i != (rear+1) ; i = (i+1) % size)
print Q[i]

S-3 If (front > rear)
i: for (i = front ; i ≤ size-1 ; i++)
print CQ[i]
i: for (i = 0 ; i ≤ rear ; i++)
print CQ[i]

S-4 Exit.

## Double ended Queue (D-Queue) ⇒

D-Queue is the type of queue or Homogeneous list of elements in which insertions & deletion operations perform on both ends, i.e we can insert elements from rear end or from the front end also; similary we can delete elements from the rear end or from the front end also. Hence it is called D-Que

(i) I/P

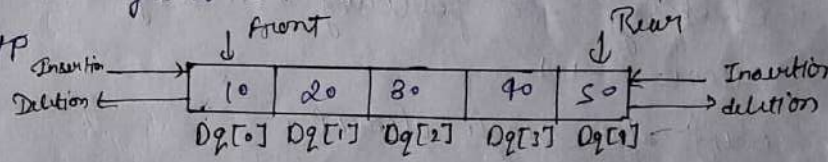



Fig: D-Queue

## Types of D-Queue ⇒

① I/P restricted D-Queue ⇒

In I/P restricted D-Queue, insertion can be done only in rear end where as deletion can be done in both rear

end & front end.

O/p restricted D-Queue ⇒

In O/p restricted D-Queue, insertion can be done from both ends (Rear & front end) where as deletion is only possible in front end.

Operations on D-Queue ⇒

(i) Insertion at Rear ends.
(ii) Deletion at front end
(iii) Deletion at rear ends.
(iv) Insertion at front end

Deletion at rear end ⇒

In this operation rear should be decremented. The following steps should be followed in deletion at rear end -

Steps - (1) Checking for ~~following~~ underflow i.e
(rear == -1)

(2) Checking for single element.
i.e if (front == rear)

There existing only single element after deleting the element i.e
(rear == front == -1)

(iii) Otherwise the rear
i.e rear = rear - 1

Insertion at front end ⇒ In this operation front value is decremented in front position the item or element is inserted:

(1) Checking for overflow.
(front == 0)

S-2 If (front == -1)
then item can't be inserted at front end

S-3 Decremented the front i.e (front = front - 1)

S-4 Put the element in ~~front position~~.

Step) Exit

Priority Queue ⇒

During To the present a priority Queue two linear arrays (2D array) are required. First array will keep the data or information and second array will keep the priority or precidence.

Structure ⇒

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 25 | 23 | 12 | 17 | 08 | 09 | 08 | ⇒ Data or Infor |
| 1 | 1 | 1 | 2 | 2 | 3 | 3 | ⇒ Priority or precidence |

* In priority Queue the information are procured as per the priority. * The lower priority information will be procured first then the higher priority.

(iii) If we want to put an information (32) as having priority 2 then it will should be

information.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 25 | 23 | 12 | 17 | 32 | 08 | 09 | 98 |
| 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 |

## Types of priority Queue ⇒

There are basically two types of priority Queue ⇒

① Ascending P.Q.

② Descending P.Q

## Ascending Priority Queue ⇒

It is a collection of items or elements in which the elements can be inserted arbitarily. But only smallest element can be removed.

'or'

In ascending P.Q. the informations get stored as per the ascending order of the priority.

Ex ⇒

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 25 | 23 | 12 | 17 | 08 | 32 | 09 | 98 | 98 | → Info^n/item |
| 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | → Priority |

## Descending P.Q. ⇒

It is collection of items. In which insertion of items can be any order but only largest element can be removed.

Or

In descending priority Queue info^n get stored as per the descending order of the priority.

Ex ⇒

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 98 | 98 | 09 | 17 | 08 | 32 | 12 | 23 | 25 |
| 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 |

## Applications of priority Queue ⇒

① Telephone director.

② Railway Ticket reservation.

③ M/c communication.

④ Sigulation moduling.

⑤ Operating system.

## Difference b/w D-Queue & Priority Queue ⇒

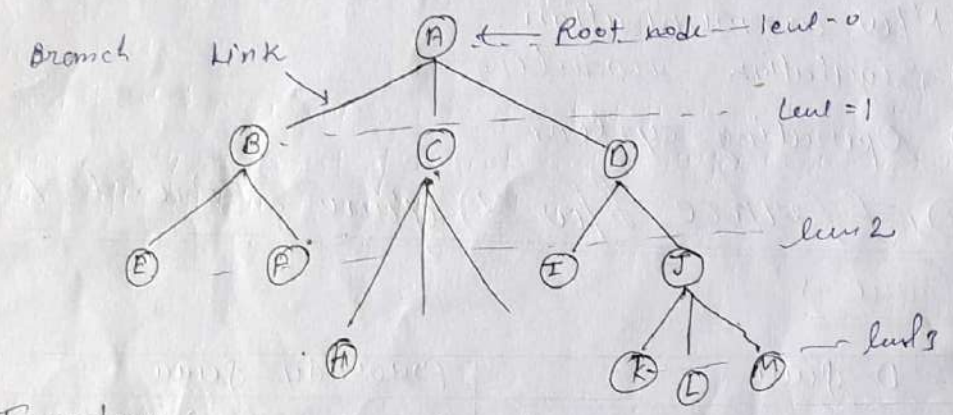
| D-Queue | Priority Queue |
|---|---|
| ① In D-Queue insertion & deletion of elements is possible from both front & Rear end. | ① In priority Queue only rear end is used for insertion & front is used for deletion of element. |
| ② In DQueue insertion & Deletion of element can be from any end (rear or front end) | ② In priority Queue element can be inserted arbitarily but while deleting an element only highest priority element can be deleted. |
| ③ | |

Scanned by CamScanner

# UNIT-3

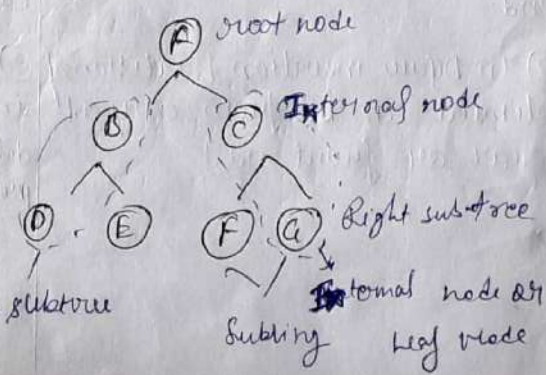
# TREE & GRAPH

**Tree ⇒** Tree is a non primitive non linear data structure In which items are arranged in a shorted sequence. It is used to represent hierarical relationship existing among several data items.



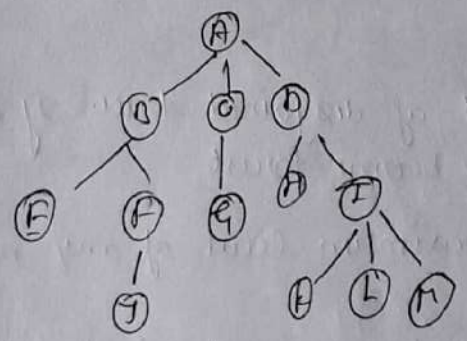

**Terminology**

1. Root node
2. Node
3. Degree of node
4. Degree of tree
5. Terminal node
6. Non terminal node
7. Sibling node
8. Level node
9. Edge node
10. path node
11. Depth node
12. Forest node




**Degree ⇒** It is the no. of subtrees or no. of child of a node in a given tree

Degree of Tree → It is a maximum no. nodes in the given tree is maximum 3.




**Terminal Node ⇒** A Node with degree zero is called a terminal node or least node.

In the above tree there are 7 terminal nodes.

7 terminal node: E, J, H, K, L, M.

**Non Terminal Node ⇒** Any node (except the root node) whose degree is not zero is called non terminal node is also called intermidiate node.

None terminal Node: B, C, D, F, I

**Sibling ⇒** The children node of a given parent node are called siblings.

**Edge ⇒** It is connecting lines of two nodes.

**Path ⇒** It is a sequence of consiquetive edges from the source to destination node.

Scanned by CamScanner

**level** → The entire tree structure is level in such a way the root node is always at level zero. Then its emidiate children at level 1 and there emidiate children at level 2 up to so on.

If a node at level 'n' then its children at 'n+1' level.

**Forest** → It is a set of disjoined tree. If you remove root node then its become forest

**Depth** → It is the maximum level of any node. In a given tree.

**Binary tree** → A binary tree is a set of finite node which is either empty or consist of a root and two disjoint binary trees called left subtree & right subtree.

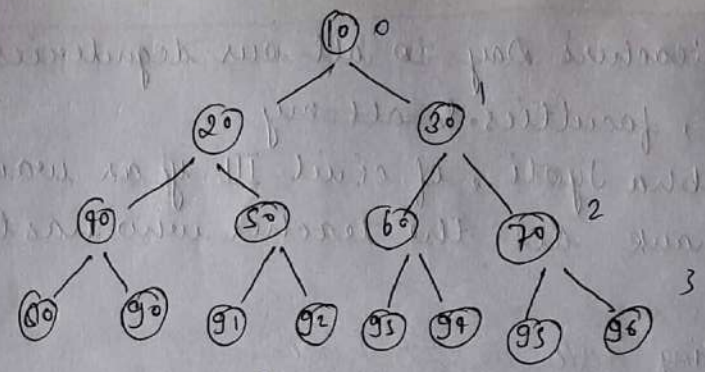
A binary tree used in a non. linear data structure the maximum degree of any node is at most two that means there may be a zero degree node or a one degree node or a 2 degree node.

**Full Binary tree** → A full binary tree is a tree in which every node has 0 or 2 children.

**Complete binary tree** → A complete binary tree is full Binary tree in which all ~~nes~~ leafs are at the same depth 'h'.

- The total no. of nodes in complete binary tree are $(2^{h+1}-1)$ where h is height (depth) of the tree.
- The no. of leaf node (n) in a complete binary tree can be found using formula $\boxed{n = 2^h}$ where n = total no. leaf nodes & h is the height of Binary tree.

n =




$h = 3$

nodes in above tree $= 2^{h+1} - 1$

$= 2^{3+1} - 1$

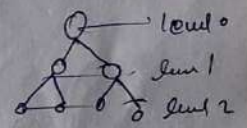
$= 15$

leaf nodes $n = 2^h$

$= 2^3$

$= 8$

Show the maximum no. of nodes in a binary tree of height h is $2^{h+1} - 1$.




Total no. of nodes (level by level)

$= 2^0 + 2^1 + 2^2 + 2^3 \ldots\ldots$

$\sum_{i=0}^{m}$

① If height = 0 then total nodes $= 2^{0+1} - 1 = 1$ node i.e root

② If height = 1 " " " $= 2^{1+1} - 1 = 3$ nodes.

Induction hypothesis.

Assume that total nodes $\leq 2^{h+1} - 1$ for full binary tree h.

$\boxed{h = K}$

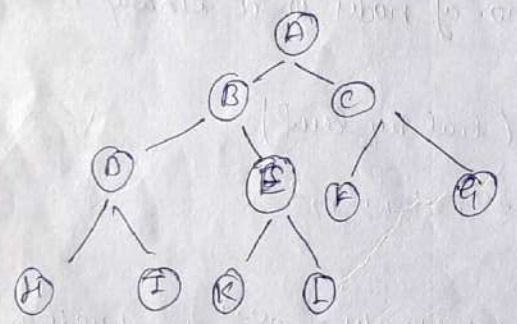
Scanned by CamScanner

Happy teacher's Day to all our deputories, HOD's, faculties. ~~& all my~~
I prabha Jyoti, of civil III year want to thank all the teacher who has taught me.

① Inserting node as I^st node-

① creating a node.
② Checking for over flow
③ putting some

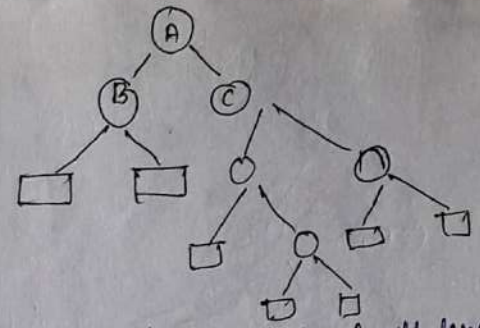
Strictly binary tree ⇒ If every non terminal node in a binary tree consists of non empty left subtree and right subtree then such a tree is called strictly binary tree.




from above fig binary tree consists of (0 & 2) degree nodes. In a strictly binary tree maximum degree of any node is either 0 degree node or 2 degree node.

Extended ⇒ The extended binary tree is said to a ~~two~~ 2-tree or an extended binary tree if in which each node will have either 0 child or 2 children. In such a case the node with two children are called or "internal nodes" & node with 0 children are called external nodes

No. of external nodes = no. of Internal nodes + 1




O: Internal
□: External

Let external path length is '$P_E$', internal path length is '$P_I$' then

$$P_E = P_I + 2 * \text{no. of internal node}$$

proof that strictly binary tree with 'n' leafs contain (2n-1) nodes

Sol^n Assume that no. of nodes in SBT will be 'm'

total no. of nodes = nodes with 0 degree + nodes with 1 degree + nodes with 2 degree.
'$n_0$' '$n_1$'s '$n_2$'

$$m = n_0 + n_1 + n_2$$

But we know in SBT there would be no 1 degree node.

$$m = n_0 + n_2 \quad \text{—— (1)}$$

In any binary tree = no. of leaf nodes ($n_0$) = nodes having 2 degree ($n_2$) + 1

i.e $n_0 = n_2 + 1$ —— (ii)

i.e $n_2 = n_0 - 1$ —— (iii)

Putting eq^n (3) in (1) we get -

$$m = n_0 + (n_0 - 1)$$

$$m = 2n_0 - 1 \quad \text{—— (iv)}$$

From eq^n (4) it is proved that SBT with n leafs contain (2n-1) nodes.

Scanned by CamScanner

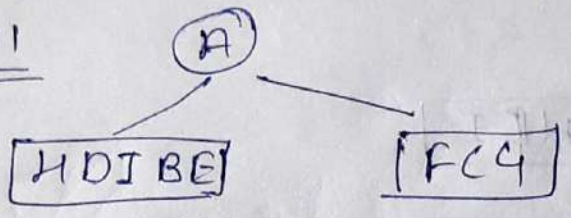
# Creation of Binary tree :-

postorder : H I D E B F G C A

Inorder : H D I B E A F C G

L ↑ R

**Step 1**

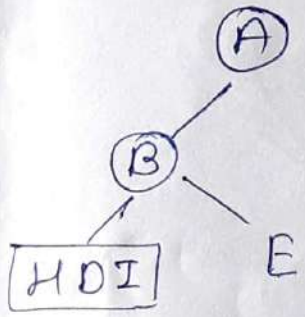



**Step 2**

post : H I D E B

inorder : H D I B E

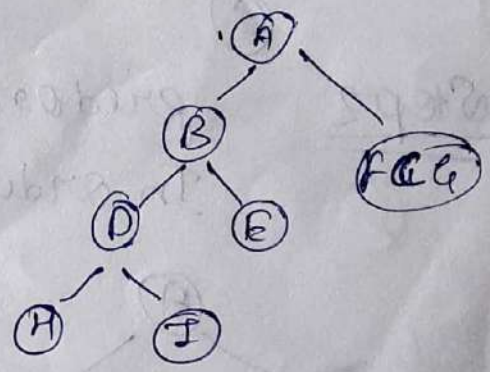
↑




**Step 3**

postorder : H I D

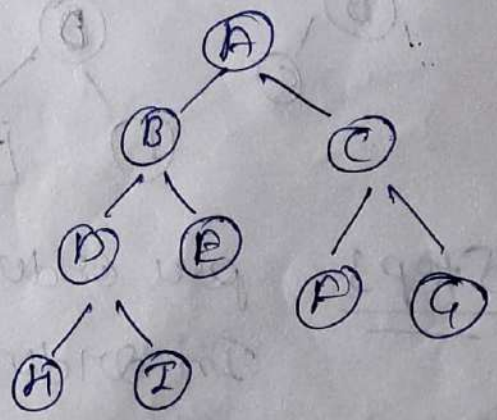
Inorder : H D I




**Step 4**

post order : F G C

In order : F C G




Kanika Verma

Kanika Verma

Kanika

Kanika Verma

Scanned by CamScanner

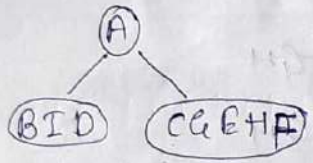
(5) i) Inorder = BIDACGEHF

Postorder = IDBGCHFEA

ii) Preorder = FAEKCDHGB

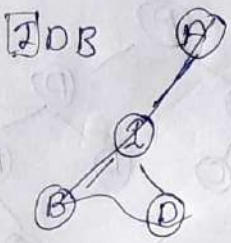
Inorder = EACKFHDBG

Step1



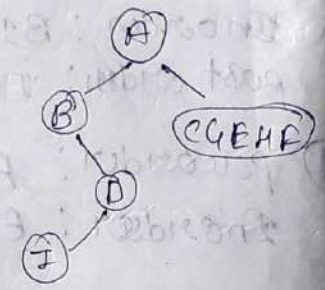

Step2 postorder = ~~BID~~ IDB

Inorder = BID



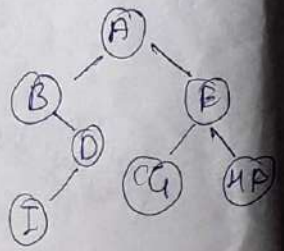

Step3 post order = ID

Inorder = ID



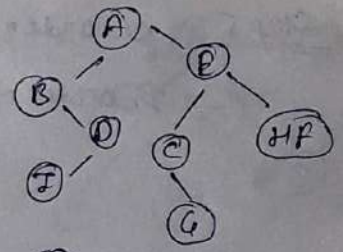

Step4 postorder = GCHFE

Inorder = CGEHF



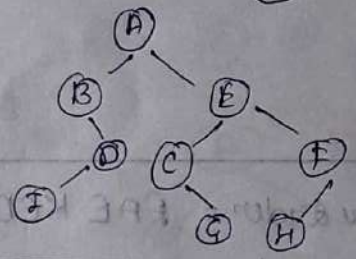

Step5 postorder = GC

Inorder = CG




Step=6 postorder = HF

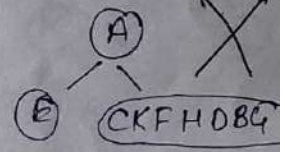
Inorder = HF




(2) Preorder = FAEKCDHGB

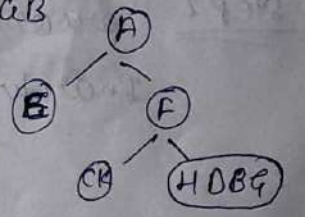
Inorder = EACKFHDBG

Step1



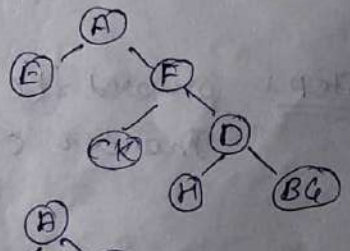

Step=2 preorder = ~~CKFI~~ FKCDHGB

Inorder = CKFHDBG



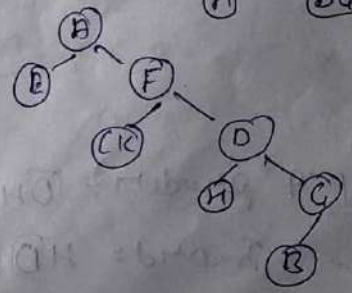

Step 3 preorder = DHGB

Inorder = HDBG



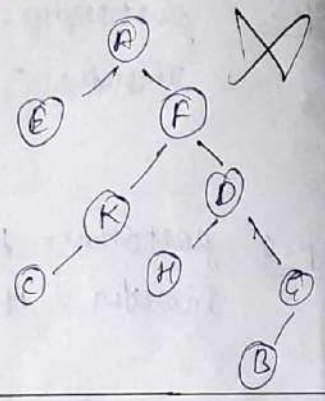

Step4 preorder = GB

Inorder = BG



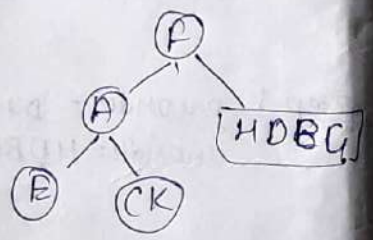

Scanned by CamScanner

Step 5 preorder = KC
Inorder = CK

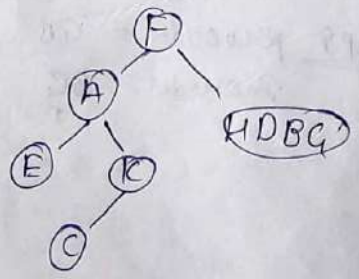



preorder = FAEKCDHGB
Inorder = EACKFHDBG

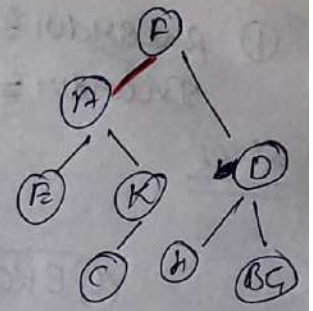
Step 2 preorder = AEKC
Inorder = EACK



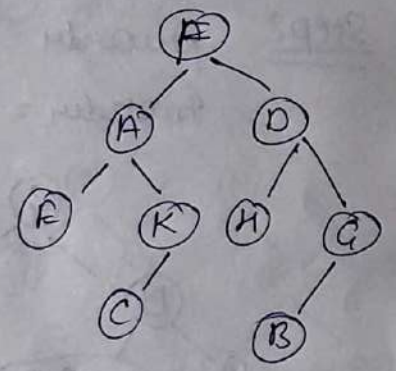

Step 3 preord = KC
Inorder = CK




Step 4 preorder = DHGB
Inord = HDBG




Step 5 = preorder = GB
Inorder = BG




① preorder = GBQACKFPDERH
Inorder = QBKCFAGPEDHR

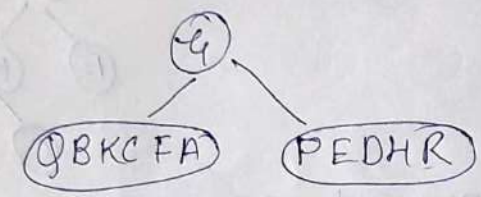
② Inorder = MEPAQTRCFK
Post = MPEQRCTKFA

③ Construct a tree for following preorder & postorder and write inorder traversal:
preorder = 29, 14, 13, 19, 17, 15, 10, 5, 8, 6, 7, 20
postorder = 13, 15, 17, 10, 19, 14, 7, 6, 20, 8, 5, 29

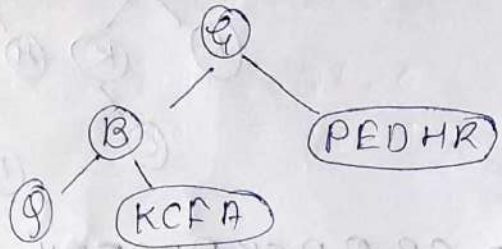
Scanned by CamScanner

① preorder = G B Q A C K F P D E R H
Inorder = Q B K C F A G P E D H R

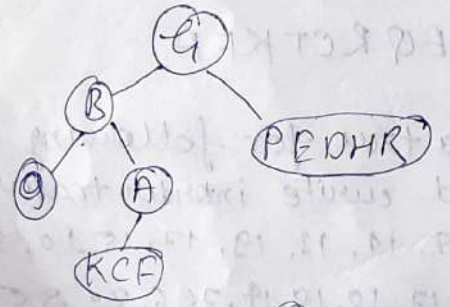
Step1



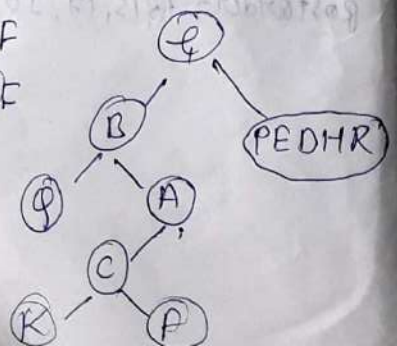

Step2 preorder B Q A C K F
Inorder = Q B K C F A



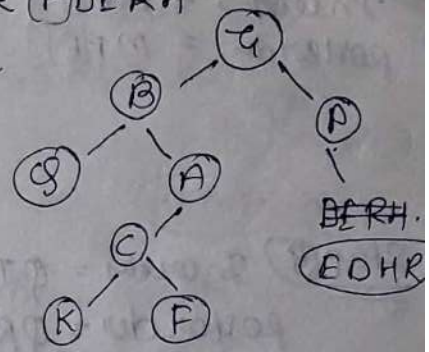

Step3 preorder = A C K F
Inorder = K C F A



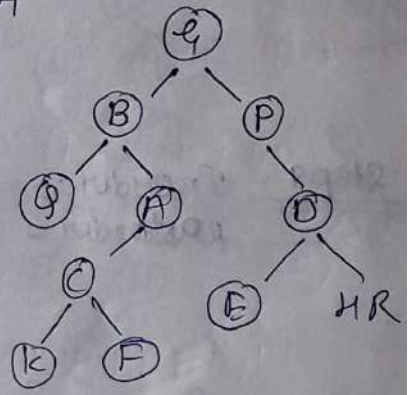

Step4 preorder = C K F
Inorder = K C F



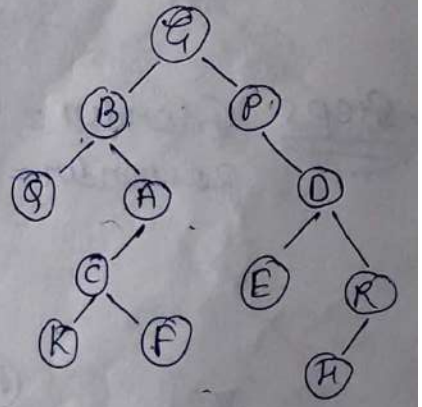

Step5 preorder = ~~PEDHR~~ P D E R H
Inorder = P E D H R




Step6 preorder = D E R H
Inorder = E D H R



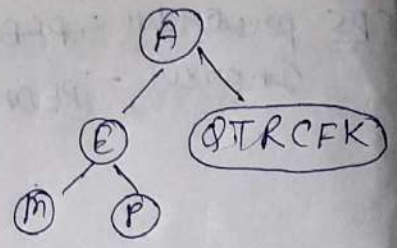

Step7 preorder = R H
Inorder = H R



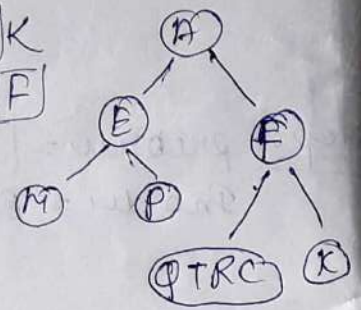

② ~~preorder = MEP~~
~~Inorder =~~
② Inorder = M E P A Q T R C F K
postorder = M P E Q R C T K F A

Scanned by CamScanner

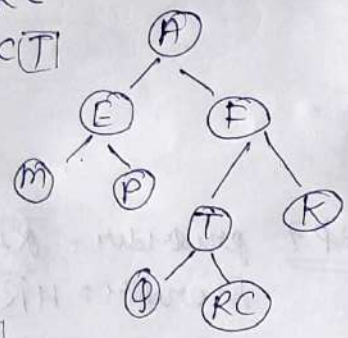
Step 2
Inorder = [M E] P
postorder = M P [E]



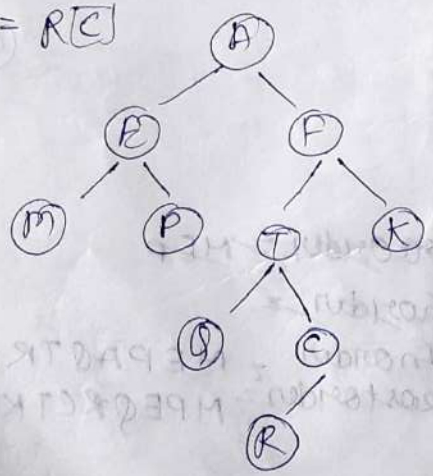

Step 3 Inorder = QTRC[F]K
postorder = QRCTK[F]




Step 4 Inorder = Q[T]RC
postorder = QRC[T]

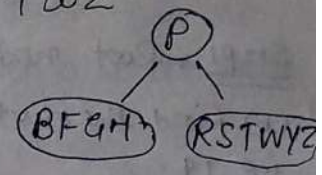



Step 5 Inorder = R[C]
postorder = R[C]

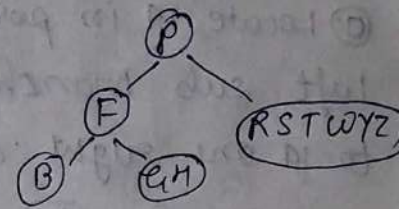



Qu Inorder = BFGH[A]RSTWYZ
Preorder = [A]FBHGSRYTWZ

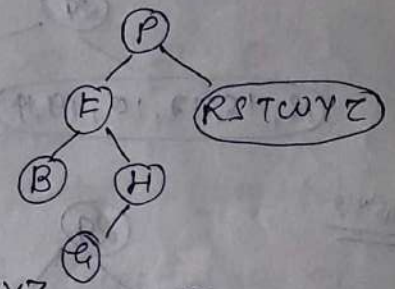
Step 1



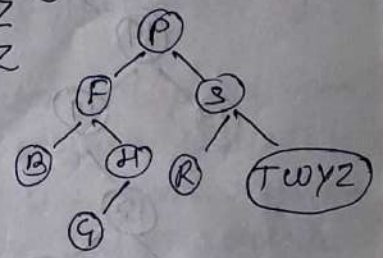

Step 2 Inorder = B[F]GH
preorder = [F]BHG



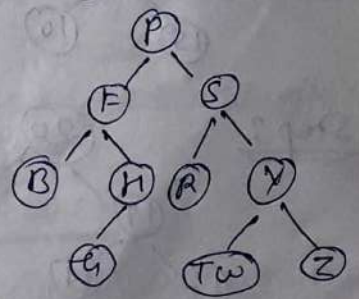

Step 3 Inorder = G[H]
preorder = [H]G



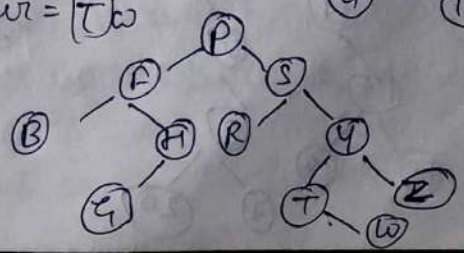

Step 4 Inorder = R[S]TWYZ
preorder = [S]RYTWZ




Step 5 Inorder = TW[Y]Z
preorder = [Y]TWZ




Step 6 Inorder = [T]W
Preorder = [T]W



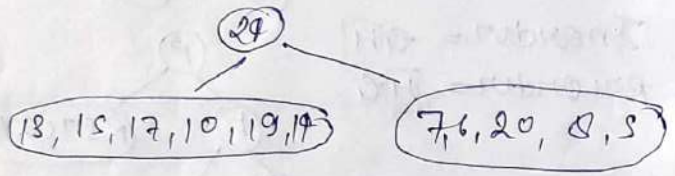

Scanned by CamScanner

Qu preorder = 24, 14, 15, 19, 17, 13, 10, 5, 8, 6, 7, 20
Inorder = 13, 15, 17, 10, 19, 14, 7, 6, 20, 8, 5, 24

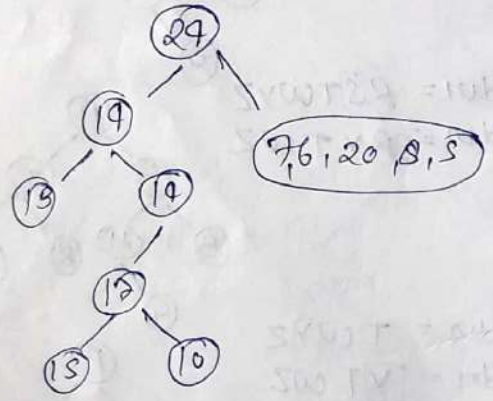
Step 1 Root node : 24

(b) find the next adjacent value of root node i.e 14.

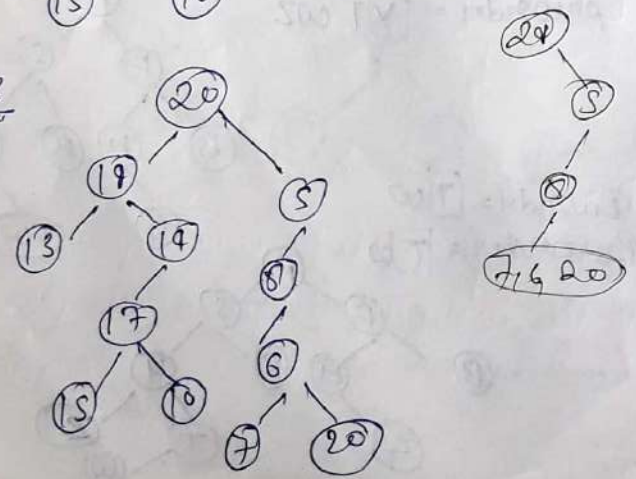
(a) Locate 14 in post order. All the value left sub branch all the values right to 14 one right sub branch




Step 2



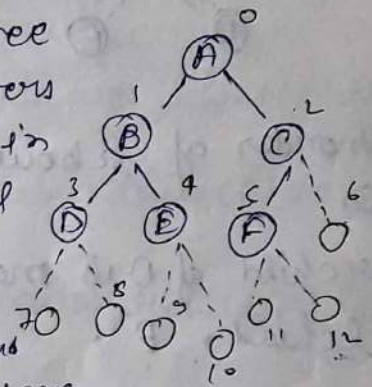

Step 3




## Threaded Binary Tree →

In a linked list representation of binary a tree tho no. of null links (null pointers) are actually more than NOR null pointers.

Eg ⇒ In given binary tree there are 7 null pointers (no. 6 to 12) and shown in doted lines. And 5 actual pointers (no. 0 to 5), In all there are 12 pointers. We can generalize it that for any binary tree there will be (n+1) null pointers & (2n) total pointers




The object is to make effective use of these null pointers.

A. J Perlis & C. Thronton proposed an idea to make use of these null pointers.

According to this idea we can replace all the null pointers by appropriate pointer values called "Threads".

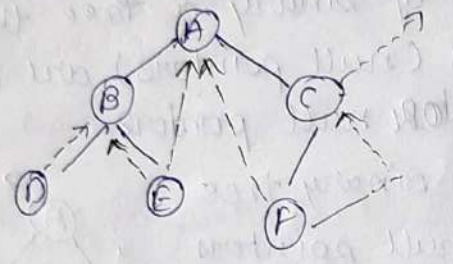
A binary tree is threaded accordingly to a particular traversal method.

## Null pointer replaced scheme ⇒

⇒ If The left link (L child/lft) of a node p is null then this null is replace by the node which imidiately preceds node in order traversal.

⇒ If the right link (R child) of a node

p is null then this link replace by a pointer to its inorder successor node




Inorder of above binary tree: DBEAFC

So

rchild of D is made to point to B.

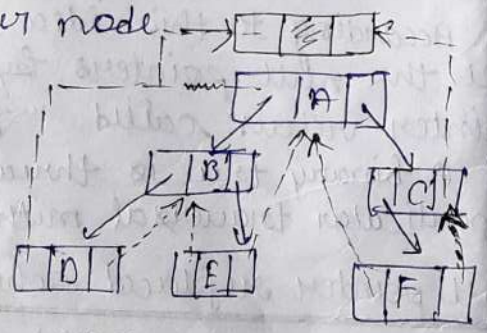
lchild " E " " " " B.

rchild " E " " " " " A.

l child of F " " " " " A

r child of " " " " " C.

⇒ Two Dangling pointers such as left child [d] and rchild [C] are made two point a header node.




In memory Representation of TBT it is necessary to two distinguish b/w in a normal pointer & thread.

| lthread | lchild | Data | rchild | rthread |
|---|---|---|---|---|
| | | | | |

{ for any node 'p' a TBT are:

* l·thread (P) = 1, indicates left child (p) is a normal pointer.
* l thread (P) = 0, indicates lchild (P) is a thread.
* r thread (P) = 1, indicates rchild (P) is a normal pointer.
* r thread (P) = 0, indicates rchild (P) is a thread.

In order traversal in TBT ⇒

* In order traversal is a technique in which left node is visited first, then its parent node and finally the right node is visited
* In the TBT the thread field is used for traversing.
* For instant in left thread (lth) and right thread (rth) value is one then that means corresponding left or right child is parent for that node. But If either of the value is zero then we have to move successor node using corresponding thread pointer.

Algo TB inorder (T)

{ head = T;

Scanned by CamScanner

```
while (1) :
{
 T= Incoder
```

```
Incoder - succ(x)
 {
  p = rchild(x)
 if (thread==1)
  }
 while (lthread ==1)
  {
   p =
```

Technics for concerting an expression into a Binary tree ⇒

Binary reprentation are tew types-

① Algebric expr. (Numbers, variables)

② Boolean expr. (True & false)

⇒ Dived & conquer used to concert Binary tree

⇒ The 'Leaf' of Binary tree are operands all other remaining nodes are 'operator'.

Ex ⇒ A + (B + C * D + E) + F/G

⇒ Note the order of preedence. or all expr in paranthesis are to be evaluated first

Scanned by CamScanner

## Huffman Algorithm:

Huffman algo. was developed by David Huffman. This algo. basically a coading technic for n coading data. Such an n coded data is used in data compression technics.

(i) In this coding method the data is inputed as the sequence ~~the~~ of characters. Then a table of frequency of occurance of each chacter in the data is built.

(ii) From the table of frequencies "Huffman tree" is constructed.

(iii) Huffman tree is further use for encoding each chacter so that binary encoding is obtain for given data

Defination of Huffman tree: Huffman tree is a binary tree generated using Huffman's coding. The Huffman's code is particular type of prefix code and in obtainting optimal data compression

Huffman's algo:

```
n ← |C|
Q ← C
for i ← 1 to n-1
    to z ← Allocate-node
    x ← left[z]
        Exact-min[Q]
    y ← right[z]
```

Scanned by CamScanner

Extract-min (Q)

$f(z) \Rightarrow f(x) + f(y)$

Enqueue (Q, z)

return Extract-min (Q)

<u>Qu</u> Find the huffman tree of following freq$^n$-

a: 50, b: 25, c: 15, d: 40, e: 75

$C = \{a, b, c, d, e\}$

$f(C) = \{50, 25, 15, 40, 75\}$

$n = 5$

$Q \leftarrow C$

Arrange freq ascending order

| c \| 15 | b \| 25 | d \| 40 | a \| 50 | e \| 75 |
|---|---|---|---|---|

$i \leftarrow 1$ to 4

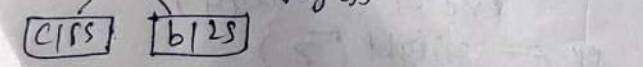
$i = 1$, $z \leftarrow$ Allocate code, $x \leftarrow$ Extract min(Q)

$y \leftarrow$ Extract min(Q)

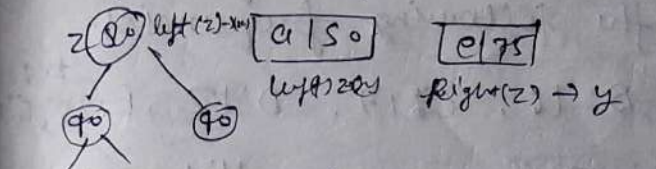
| c \| 15 | b \| 25 | d \| 40 | a \| 50 | e \| 75 |
|---|---|---|---|---|

left(z) $\Rightarrow x$ $y \rightarrow$ right(z)

$f(z) \Rightarrow 15 + 25 = 40$



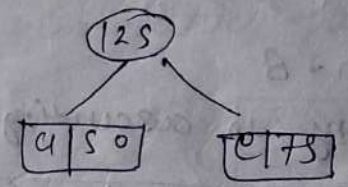

Array for $i = 2$ $f(z) = 40 + 40 = 80$




$f(z) = f(x) + f(y)$

$= 50 + 75$

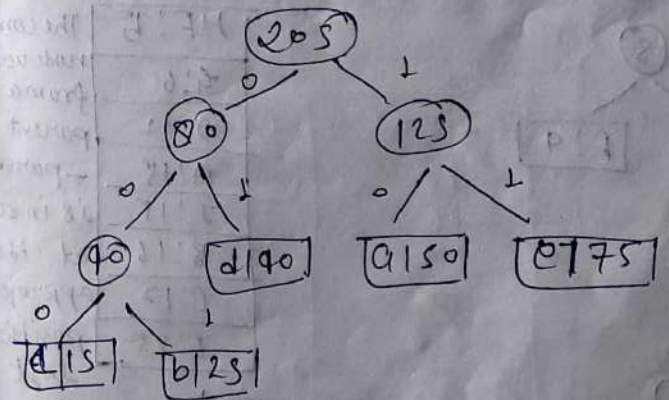
$= 125$




$f(z) = f(x) + f(y)$

$= 80 + 125$

$= 205$




<u>Unique huffman codes</u>

| Character | Frequency | Code |
|---|---|---|
| a | 50 | 10 |
| b | 25 | 001 |
| c | 15 | 000 |
| d | 40 | 01 |
| e | 75 | 11 |

Scanned by CamScanner

Draw huffman tree & generate a code for the following frequencies.

A:15, B:16, C:17, D:12, E:25, F:4, G:6, H:1, I:15

i = 7 time

C = {A, B, C, D, E, G, H, I}

f(c) = {15, 16, 17, 12, 25, 6, 1, 15}

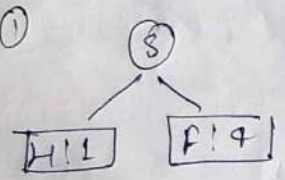
n = 8

→ Array is ascending order

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| H:1 | F:4 | G:6 | D:12 | A:15 | I:15 | B:16 | C:17 | E:25 |

⇒ We will have remove 1st two entries & create a partial Tree.

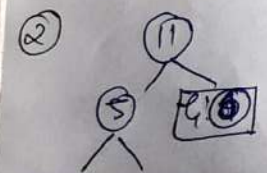
①




| |
|---|
| HF:5 |
| G:6 |
| D:12 |
| A:15 |
| I:15 |
| B:16 |
| C:17 |
| E:25 |

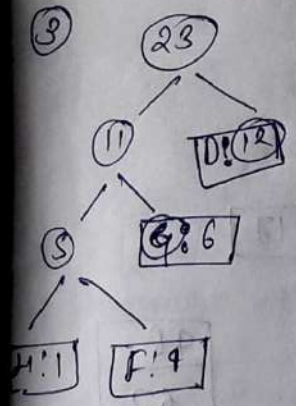
The combined node values from a new parent & this parent node is inserted at its appropriate position as left

②



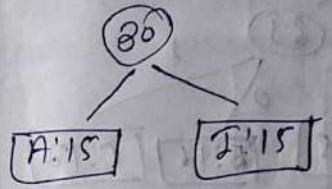

| |
|---|
| HFG:11 |
| D:12 |
| A:15 |
| I:15 |
| B:16 |
| C:17 |
| E:25 |

③



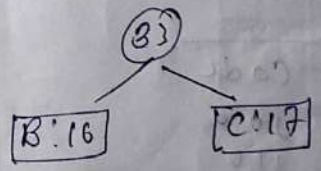

| |
|---|
| ~~HFGD:25~~ |
| A:15 |
| I:15 |
| B:16 |
| C:17 |
| 23 |
| E:25 |

④ Remove 1st two entries from Queue



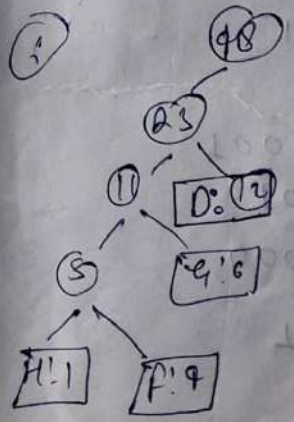

| |
|---|
| B:16 |
| C:17 |
| 23 |
| E:25 |
| 30 |

⑤ Remove 1st two entries from Queue



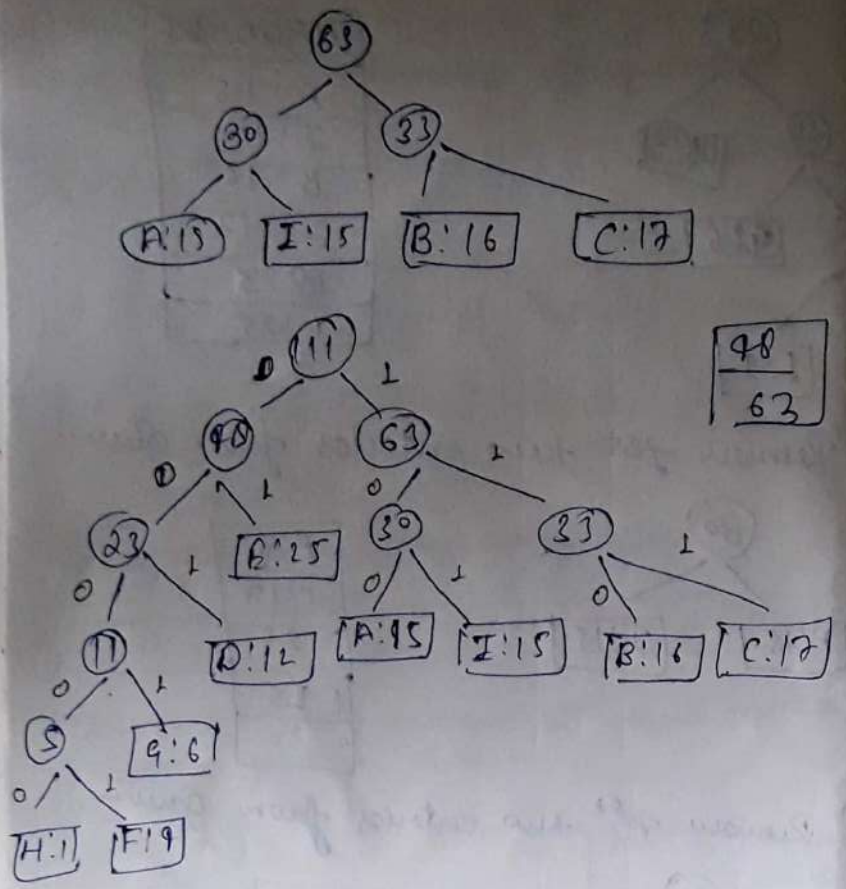

| |
|---|
| 23 |
| E:25 |
| 30 |
| 33 |

⑥




| |
|---|
| 30 |
| 33 |
| 48 |

Scanned by CamScanner




| Character | Frequency | Code |
|---|---|---|
| A | 15 | 100 |
| B | 16 | 110 |
| C | 17 | 111 |
| D | 12 | 001 |
| E | 25 | 01 |
| F | 4 | 00001 |
| G | 6 | 0001 |
| H | 1 | 00000 |
| I | 15 | 101 |

Que Create a Huffman tree of following
29, 55, 13, 67, 88, 36, 17, 61, 24, 76

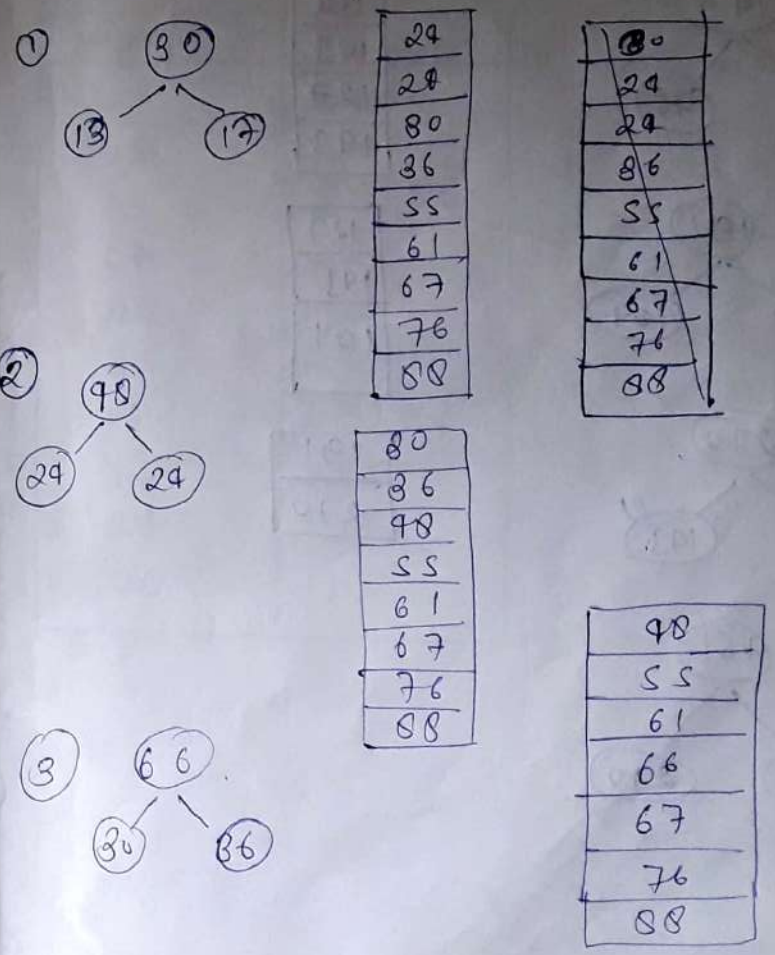
$n = 10$

$i = 9$

→ Arrange in ascending order -

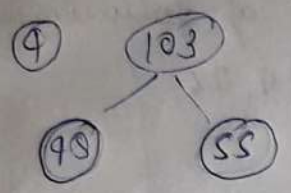
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 13 | 17 | 24 | 24 | 36 | 55 | 61 | 67 | 76 | 88 |

⇒ We will remove first two entries & create a partial tree.



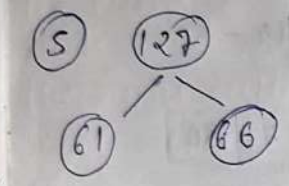

Scanned by CamScanner

(4)



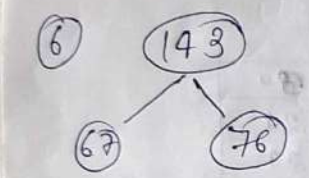

| |
|---|
| 61 |
| 66 |
| 67 |
| 76 |
| 88 |
| 103 |

(5)



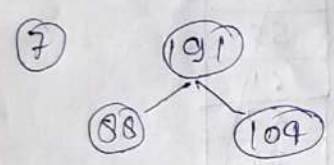

| |
|---|
| 67 |
| 76 |
| 88 |
| 103 |
| 127 |

(6)



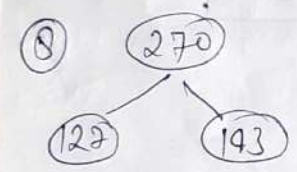

| |
|---|
| 88 |
| 103 |
| 127 |
| 143 |

(7)



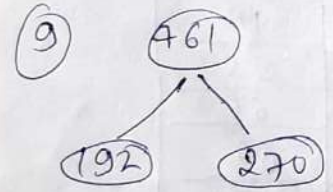

| |
|---|
| 127 |
| 143 |
| 191 |

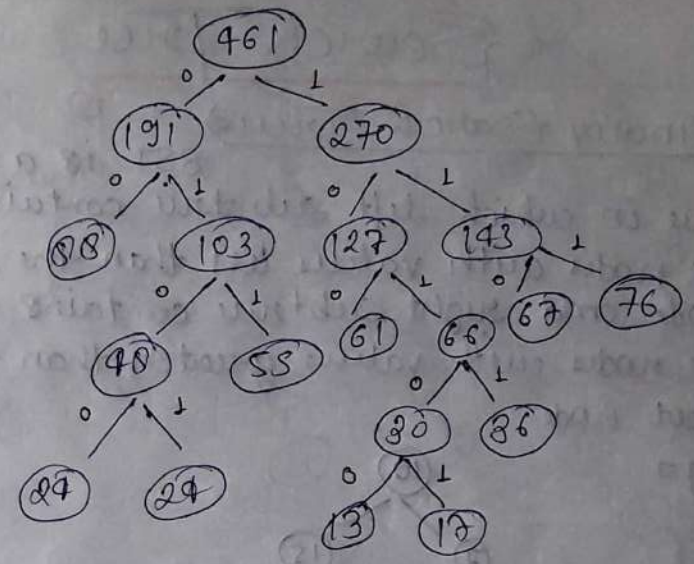
(8)




| |
|---|
| 191 |
| 270 |

(9)




| Number | Code |
|---|---|
| 24 | 0100 |
| 55 | 011 |
| 13 | 10100 |
| 67 | 110 |
| 88 | 00 |
| 36 | 1011 |
| 17 | 10101 |
| 61 | 100 |
| 24 | 0101 |
| 76 | 111 |

88
109
191

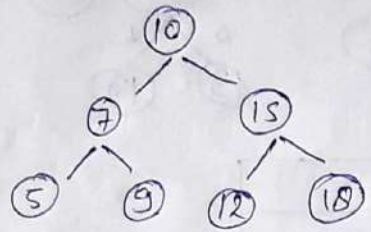
Scanned by CamScanner

# Search Tree

## Binary Search Tree ⇒

BST is a Binary Tree in which left sub tree contains all the nodes with values less than the root node and right subtree contains all the nodes with values greater than the root node.

Eg ⇒




* The binary search can be represented as an abstract data type (ADT).

Abstract data type BST ⇒

**Instances ⇒**

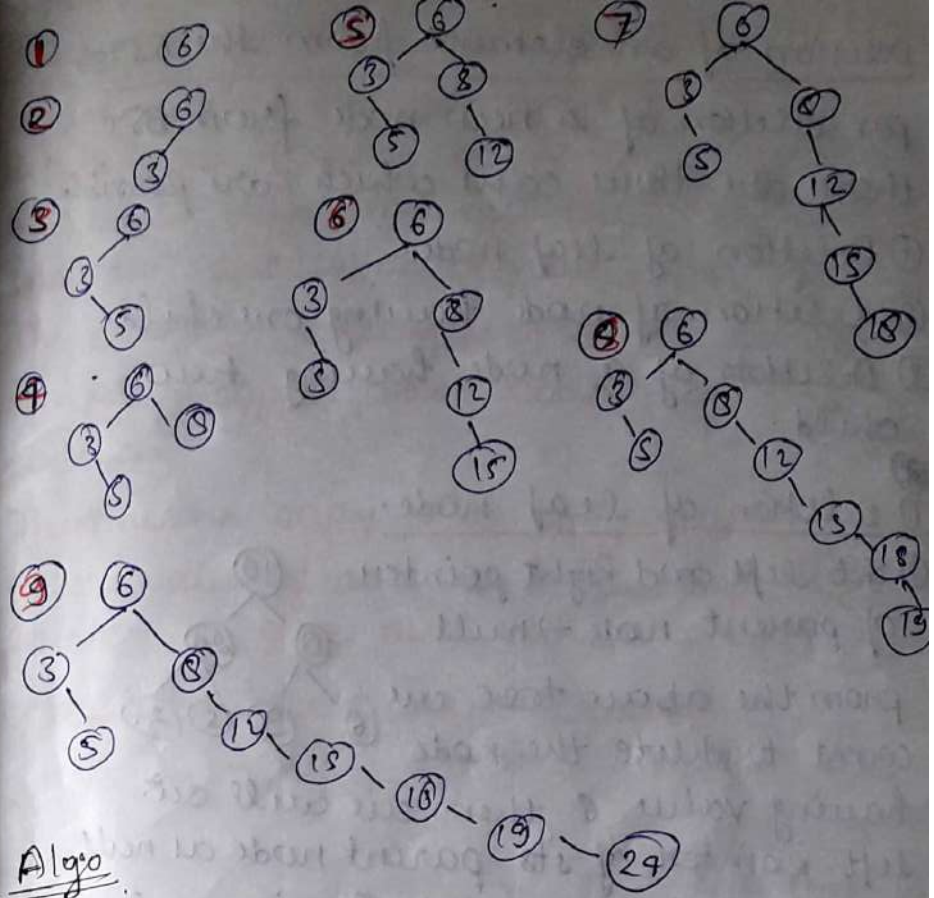
BST is a Binary tree in which the value of left child is less than root node and value of right child is greater then its parent node.

**Operations ⇒**

(i) Create (ii) Display (iii) Insert (iv) Delete (v) Search.

**Insertion & Deletion of BST ⇒**

Eg ⇒ 6, 3, 5, 8, 12, 15, 18, 19, 20, 24




**Algo**

① Read the value for the node which is to be created and store it in a node called 'New'.

② Initially If (root != Null) then root = new

③ Again read the next value of node created in 'New'

④ If (New → value < root → value) then attach new node as a left child of root otherwise attach new node as right child of root.

⑤ Repeat step ③ & ④ for constructing required BST completly.

Scanned by CamScanner

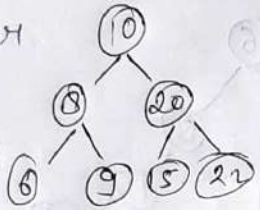
## Deletion of an element from the BST ⇒

for deletion of a new node from BST there are three cases which are possible

① Deletion of leaf node.
② Deletion of node having one child
③ Deletion of a node having two child.

### ① Deletion of leaf node

① Set left and Right pointer of parent node → null

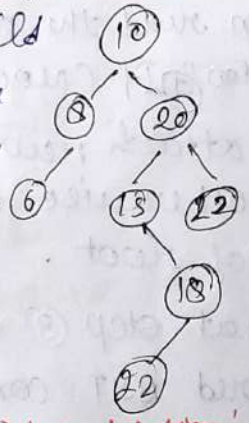


from the above tree we want to delete the node having value 6 then we will set left pointer of its parent node as null i.e left pointer of node having value 8 is set to null.

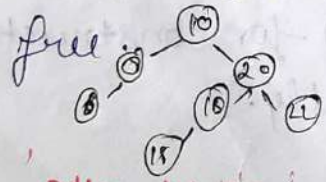
### ⑪ Deletion of node having one child ⇒

Deletion of having one child

If we want delete the node i.e 15, then we will simply copy node i.e 18 at place of 15 and then set the node free.



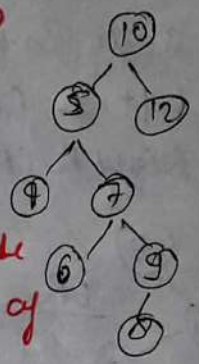
'Before deletion'



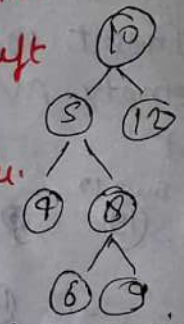
'After deletion'

### ③ Deletion of a node having two children

Let us consider that we want to delete node having value 7, we will then find out the inorder successor of node 7. The inorder successor will be simply copied ~~had~~ at location of node 7.



That means copy 8 at the position where value of node is 7. Set left pointer of 9 as null. This completes the deletion procedure.

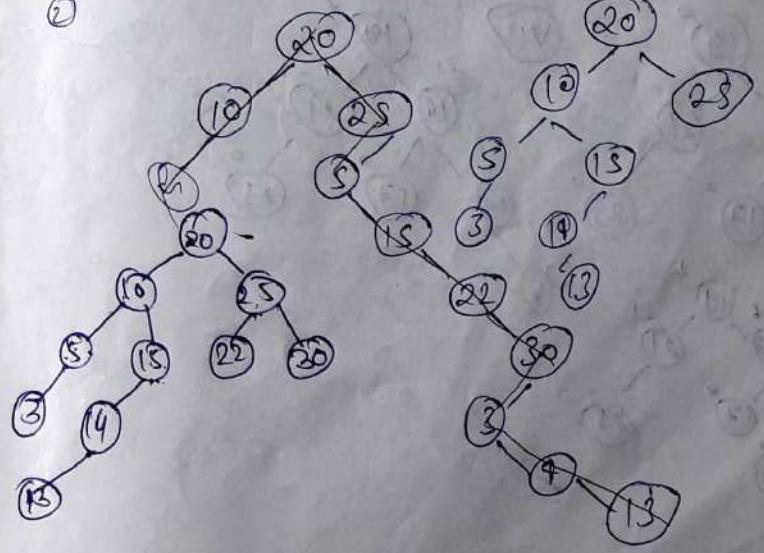


'After deletion'

**Ques** Create BST for following data.

20, 10, 25, 5, 15, 22, 30, 3, 14, 13

②



Scanned by CamScanner

## Types of Rotation

Single Rotation

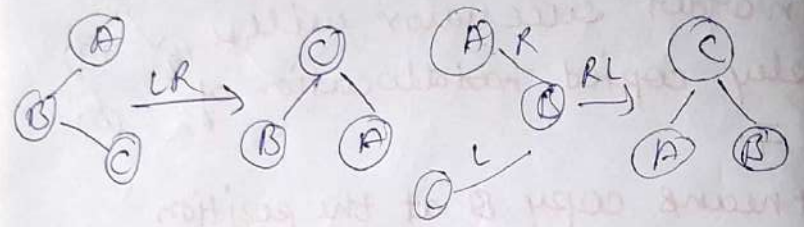
① Left L (LL)

② Right R (RR)

Double Rotation

① LR

② RL ⇒ LR.



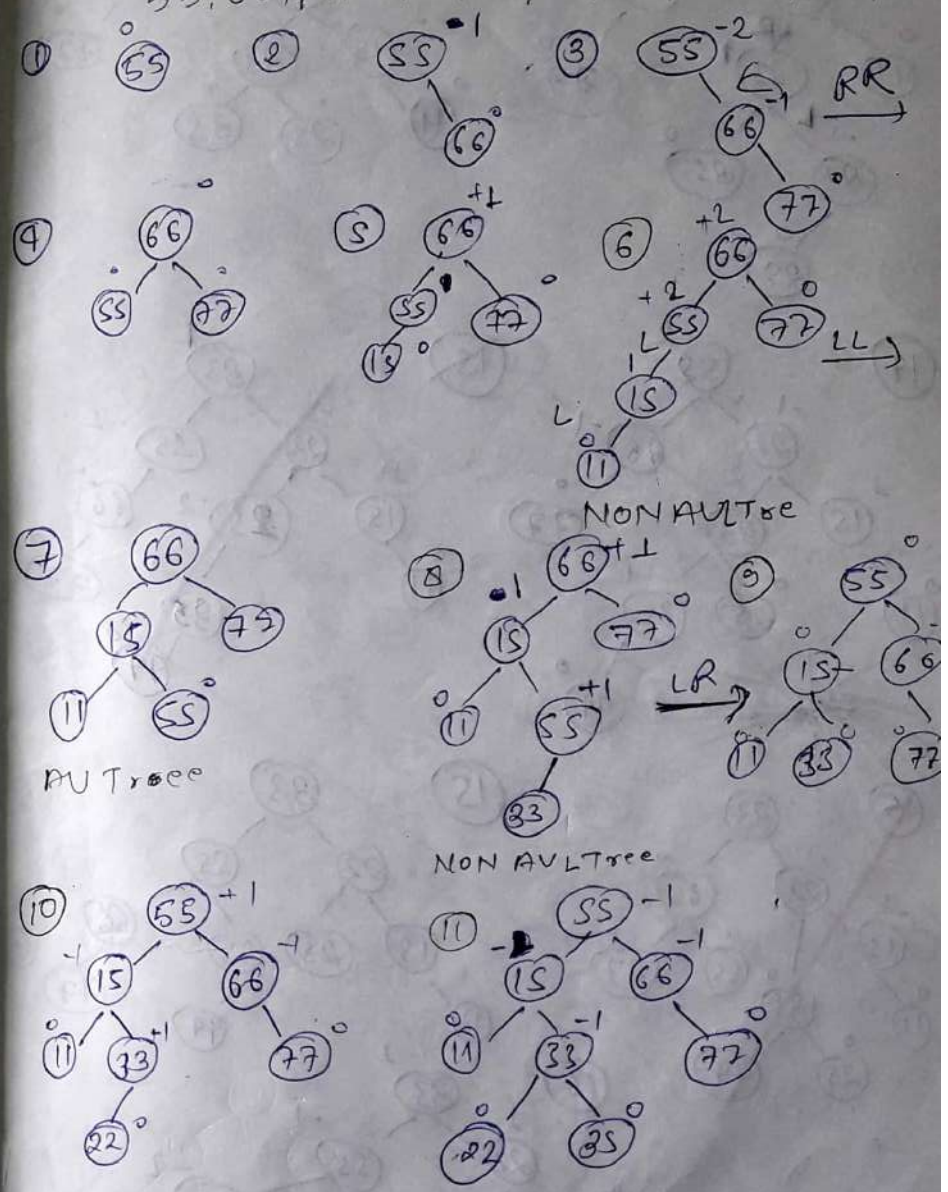

Q. Insert 19, 16, 21, 11, 17, 25, 6, 13 in and empty AVL Tree.

Ans

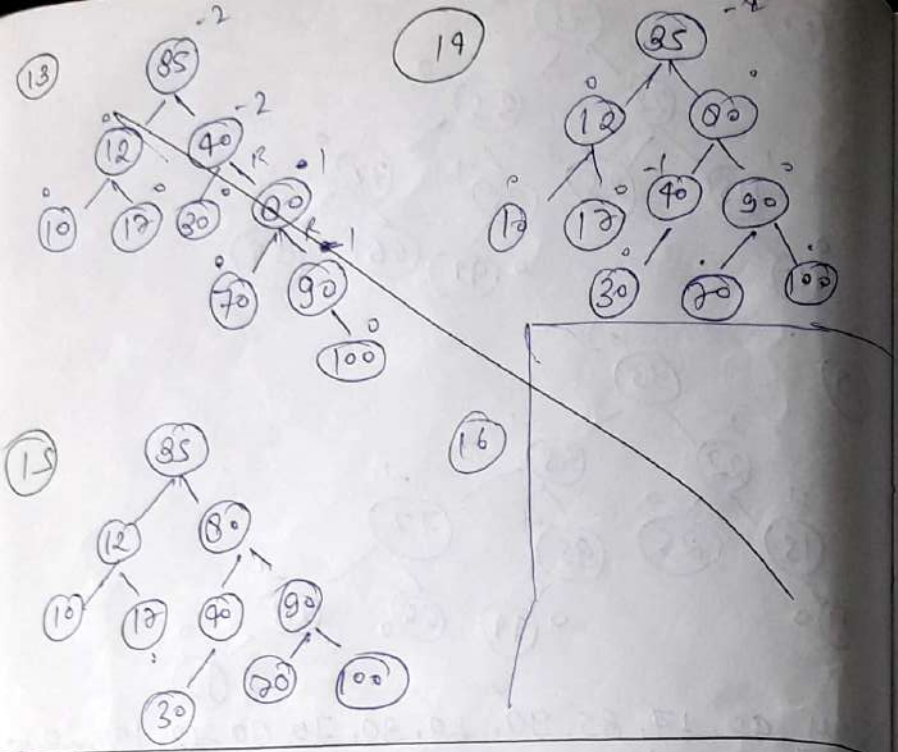
Construct a Right balance of tree of following sequence:

RL=LR
LL=R
RR=L

55, 66, 77, 15, 11, 33, 22, 85, 25, 44, 88, 99

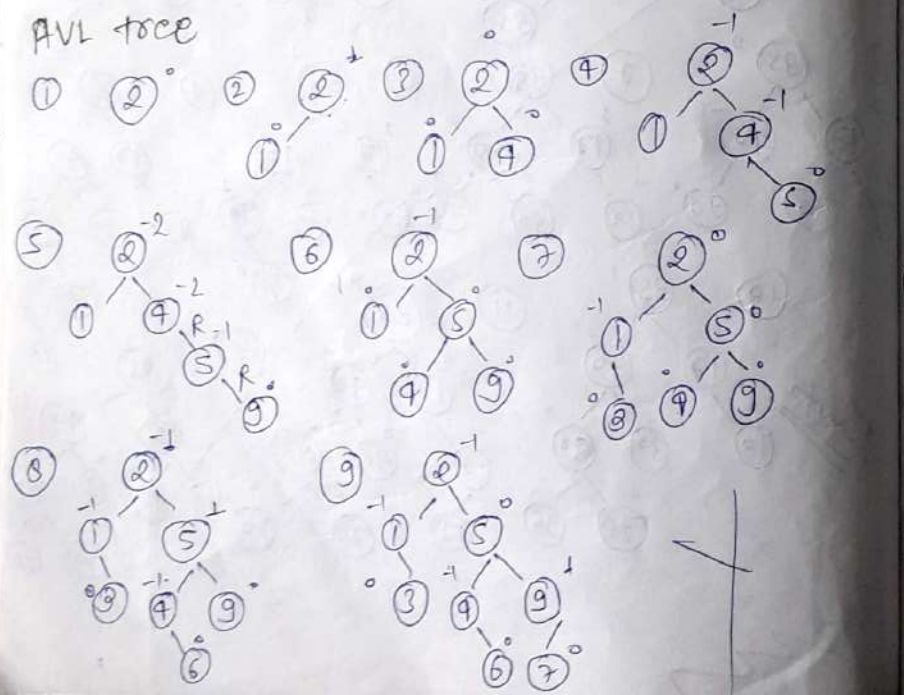



Scanned by CamScanner

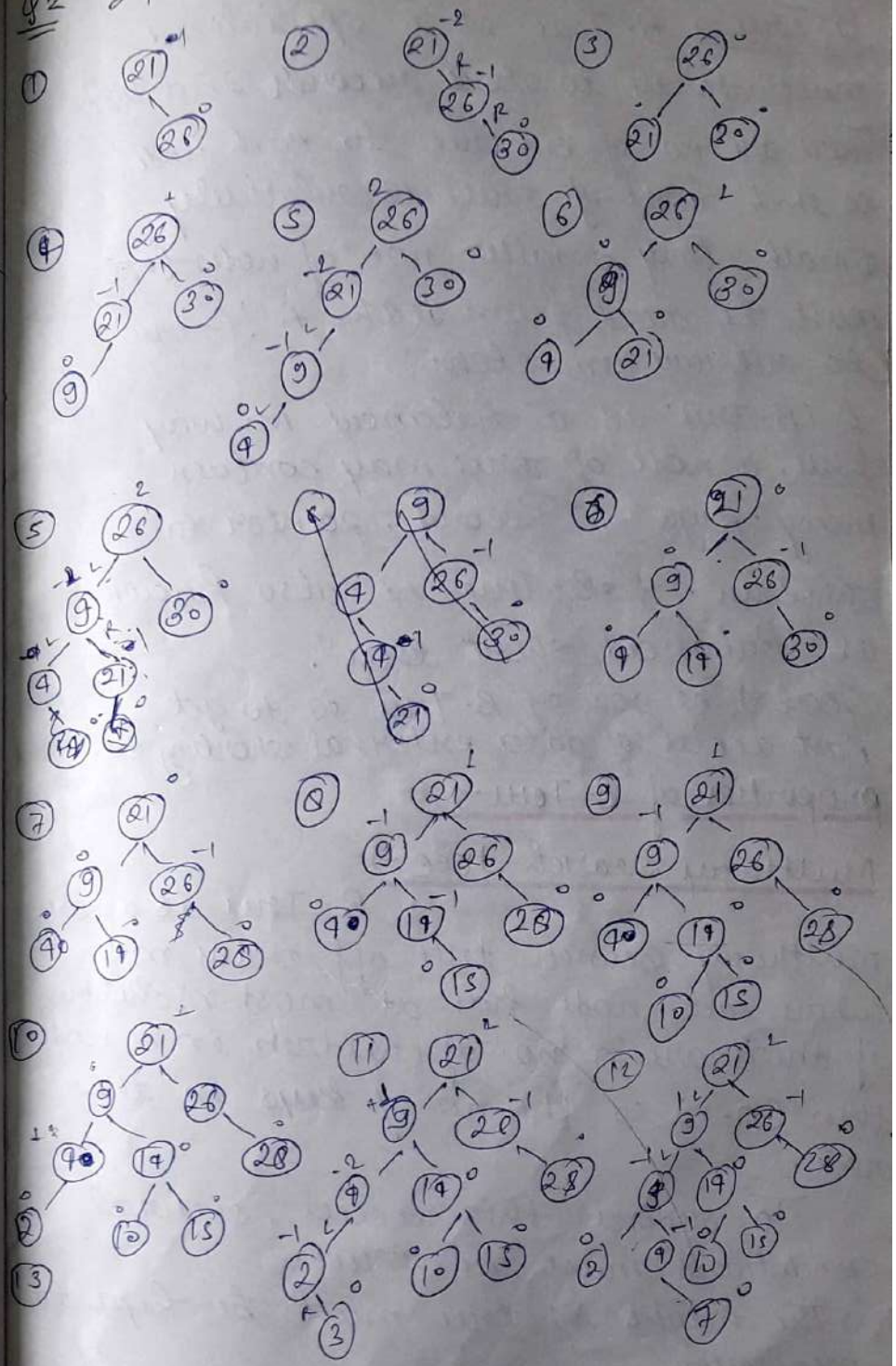


Qus 1. Show the

2, 1, 4, 5, 9, 3, 6, 7 into an empty AVL tree



Q2 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7



Scanned by CamScanner

Complexicity of AVL Tree ⇒ O(log n)

B-Tree ⇒ B-Tree is a specialized multivate use to store records in a disk. There are no. of subtree to each node, so that height of tree, is relatively small. Only smaller no. of nodes ~~that~~ must be read from disk A 'B-Tree' to retrieve an item.

A 'B-Tree' is a balanced M-way tree, a node of tree may contain many keys & Record, pointer to children. A B-Tree is also known as "Balanced short tree".

Goal ⇒ The goal of B-Tree is to get fast access of data external shorting etc.

~~Properties of B-Tree ⇒~~

Multiway search ~~tree~~ ⇒

B-Tree is a way multiway search tree of order 'm' where each node has at most 'n' children if there are 'n' no. of children in a node then (n-1) is the no. of keys in the node.

To reduce this access, several conditions must be true.

(i) The height of tree must be kept to minimum.

(ii) There must be no empty subtrees above the leafs of the tree.

(iii) The leafs of the tree must all be on the same level.

(iv) All nodes except leafs must have atleast some minimum of children.

Properties of B-Tree ⇒

① Each node has maximum of 'M' children and a minimum of 'M/2' children or any no. from 2 to the maximum.

② Each node has one fewer keys that children with a maximum of (M-1) keys

③ ~~The~~ Keys are arranged in a defined order with the node. All keys in the subtree to the key of left are predecessor of the key and that on the right are successor of the key

④ When a new key is to be inserted into a full node, the tree is split into 2 nodes and the key with median value is inserted in the parent node. In case the parent node is the root, a new root is created.

⑤ All leafs on the same level

B-Tree Insertion ⇒

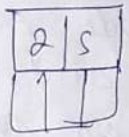
Scanned by CamScanner

Q. Construct B-Tree to insert following elements.
Order of tree is 3 -

5, 2, 13, 3, 46, 72, 4, 6, 9, 22

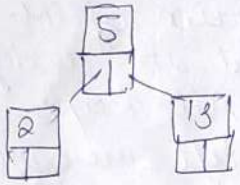
Sol^n Key = 3-1 ⇒ 2 keys are allowed.

Step1 Insert 5, 2

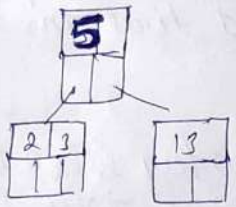



Step2 Insert 13

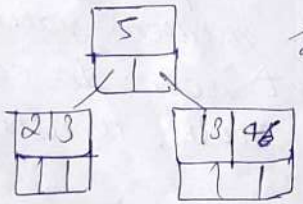
seq. 2, 5, 13. The 5 will split.




Step3 Insert 3

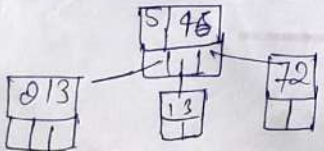



Step4 Insert 46



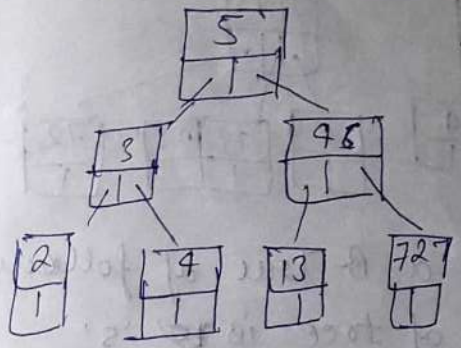

Step5 Insert 72

seq. 13, 46, 72 i.e 46 will go up.

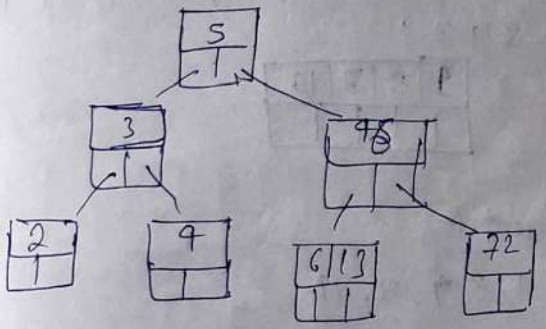



Step6 Insert 4

Seq. 2, 3, 4 → split then seq. 3, 5, 46 then again split i.e 5 will go up.

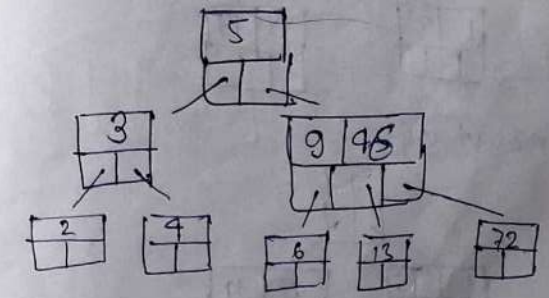



Step7 Insert 6



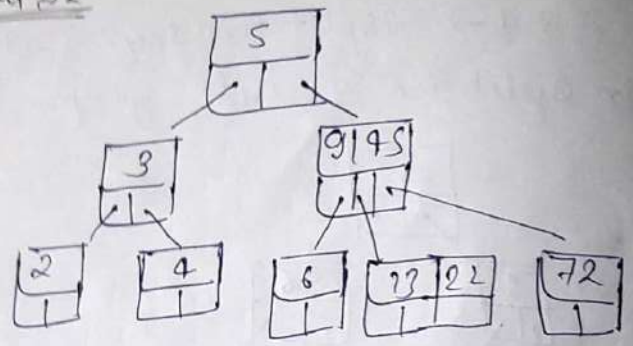

Step8 Insert 9

Seq. 6, 9, 13 i.e 9 will go up.




Scanned by CamScanner

Insert 22

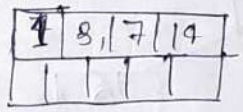



Q. Construct a B-Tree of following data if order of tree is 'S'

3, 19, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 10, 18, 24, 25, 19

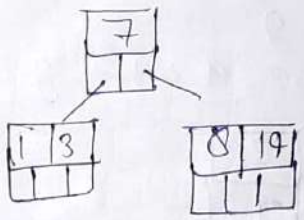
Sol$^n$ Insert 3, 19, 7, 1

Step 1

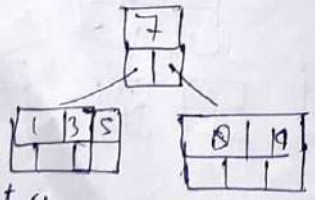



Step 2 Insert 8

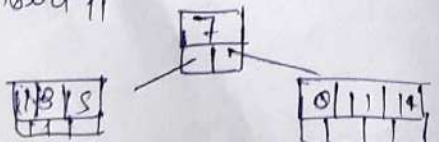
Seq = 1, 3, 7, 8, 19



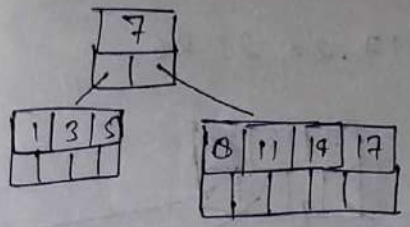

Step 3 Insert 5




Step 4 Insert 11

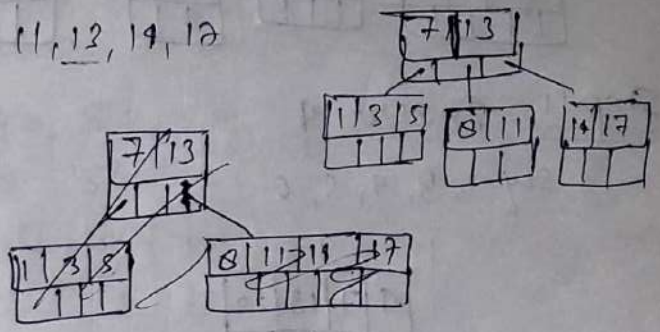



Step 5 Insert 17

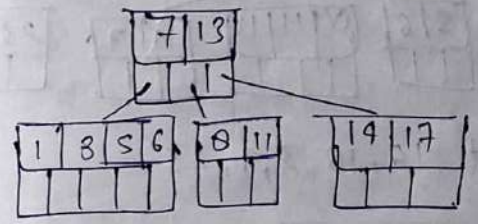



Step 6 Insert 13

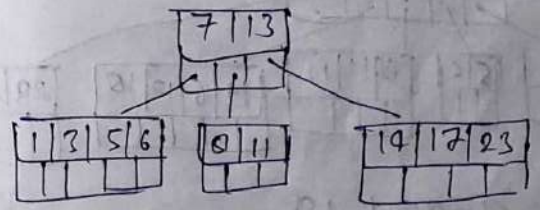
Seq = 8, 11, 13, 19, 17



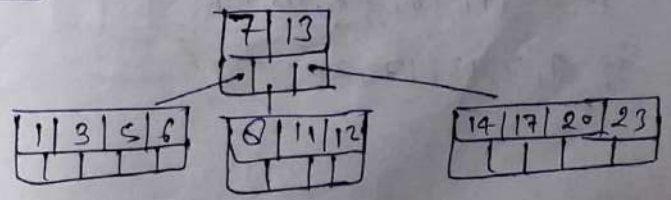

Step 7 Insert 6




Step 8 Insert 23



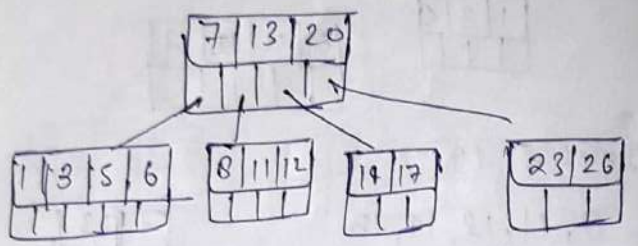

Step 9 Insert 12, 20




Scanned by CamScanner

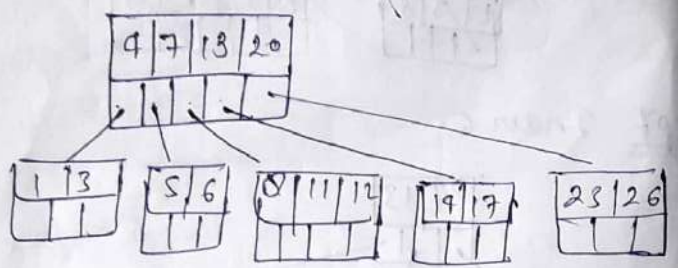
**Step 10** Insert 26

Seq = 10, 17, 20, 23, 26

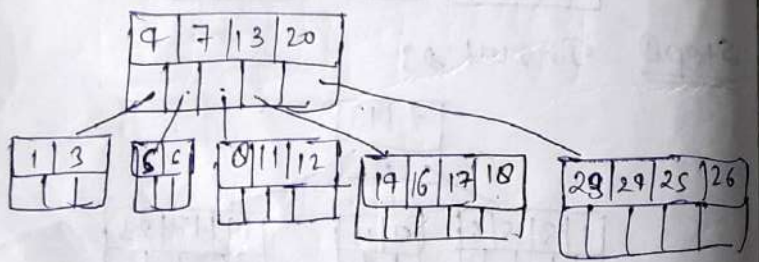



**Step 11** Insert 4

Seq = 1, 3, 4, 5, 6



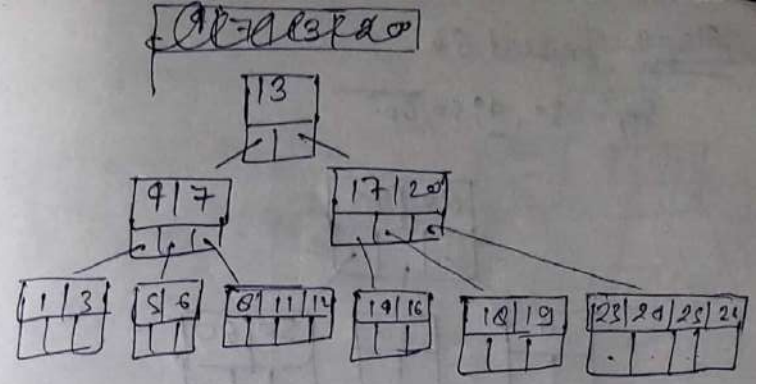

**Step 12** Insert 16, 18, 2, 9, 25




**Step 13** Insert 19

Seq 14, 16, 17, 18, 19

Seq 4, 7, 13, 17, 20




① Construct a B-Tree.

10, 20, 30, 40, 50, 60, 70, 80, 90, if order of tree is '3'.

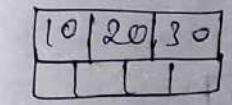
② Construct a B-Tree of following data if order of tree is '4' & '3'.

20, 40, 100, 70, 90, 110, 130, 5, 17, 120, 50, 80, 6

③ F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B if B-Tree of order '5'.

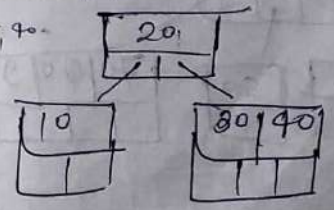
④ Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E H, S, K, N, R, G, Y, F, O, V order of '4'

① **Step 1**

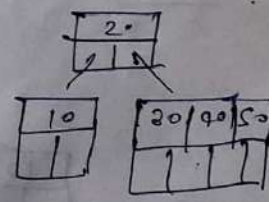



**Step 2** Insert 40

Seq 10, 20, 30, 40



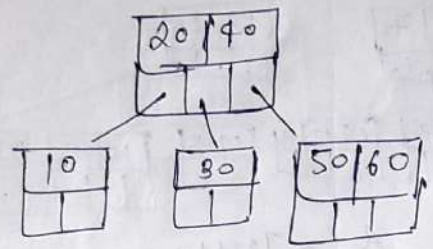

**Step 3** Insert 50




Scanned by CamScanner

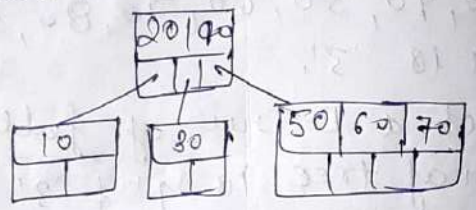
**Step 4** Insert 60

Seq. 30, 40, 50, 60



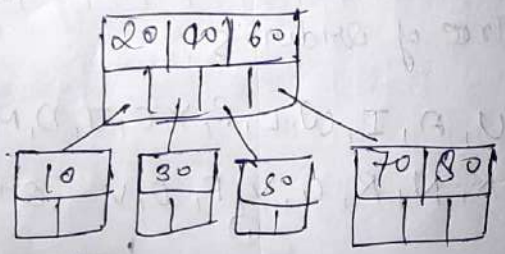

**Step 5** Insert 70

Seq

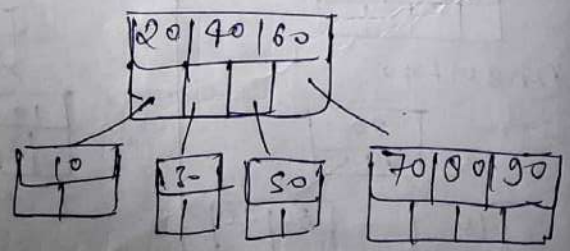



**Step 6** Insert 80

Seq 50, 60, 70, 80



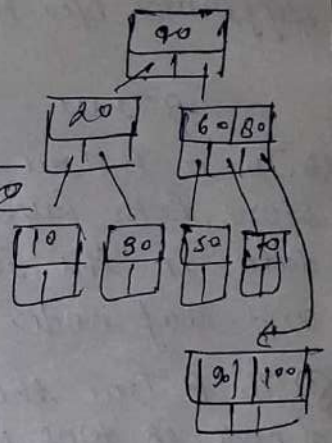

**Step 7** Insert 90




**Step 8** Insert 100

Seq 70, 80, 90, 100

Seq 20, 40, 60, 80




## Application of B-Trees

① B-Tree are used in implimenting the complex data base

② It is are used in indexing on non-primary fields.

③ External Shorting.

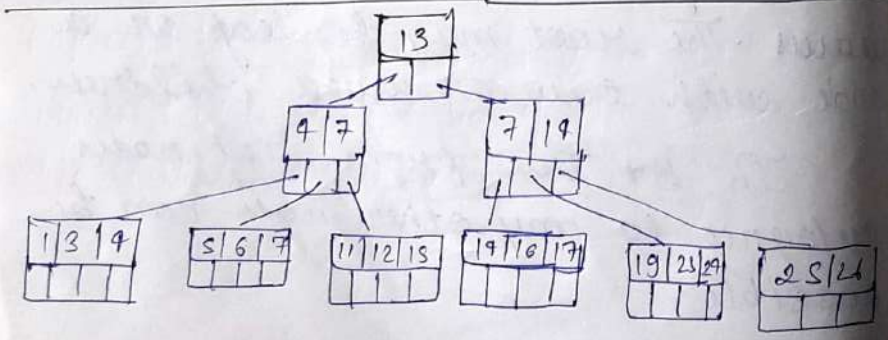
**B + Tree ⇒** It is n-array tree with a variable but open often large no. of children per node. A B+ Trees consist of a root, internal node & leaves. The root may be leaf or a node with two or more children.

In B+ Tree from leaf nodes reference to any other node can be possible

Scanned by CamScanner

## Difference b/w B Tree & B+ Tree ⇒

| B Tree | B+ Tree |
|---|---|
| ① In a B Tree you can store both keys and data in the internal and leaf node | ① In a B+ Tree you have to store data in the leaf nodes only. |
| ② In B Tree the traversing of node is done in inorder manner. | ② There is no need to traverse the tree in inorder manner. |
| ③ In B Tree the leaf node can not store using link list. | ③ In this tree leaf node data are arrange sequencial link list. |
| ④ These trees waste space | ④ These trees don't waste space. |
| ⑤ B-Tree is time consuming | ⑤ B+ Tree is not time consuming because it provide faster key access to any key |




## Difference between AVL Tree & BST Tree

| AVL Tree | BST Tree |
|---|---|
| ① AVL Tree is a height balance search tree | ① BST is not a balance tree. |
| ② There is a requirement of computation of balance factor each node in an AVL Tree. | ② There is no concept of Balance Factor. |
| ③ Searching of any desired node is faster due to balancing of height. | ③ Searching is not efficient when there are large no. of nodes in a tree. |
| ④ It is complex to implement | ⑤ Simple to implement. |

**Que** What is min. no. of nodes in an AVL Tree with hight 'S'

**Solⁿ** Let 'h' height of an AVL Tree.

$$N(h) = 1 + N(h-1) + N(h-2)$$

① If there is only 1 node then height will be '0'. similarly if there are.

Scanned by CamScanner

# UNIT 4 — GRAPH

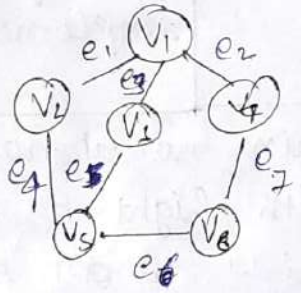
**Graph ⇒** Graph is collection of two sets 'V' & 'E', where V is a finite non-empty set of vertices and E is a finite non empty set of edges. Hence we can define following relation

$$\boxed{V(G) = (V, E)}$$

⇒ vertices:- It is nothin but the nodes in graph

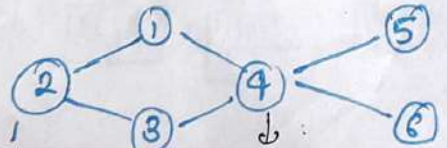
⇒ Two adjacent vertices are joined by 'edges'.

Ex ⇒




## GRAPH TERMNIOLOGY ⇒

**Adjacent Vertices ⇒** Vertices $V_1$ is said to be adjacent to a vertex $V_2$ if there is an edge $(V_1, V_2)$ or $(V_2, V_1)$



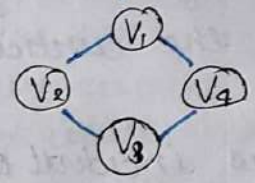

Vertices (1, 3) vertices (1, 3, 5, 6)

**path ⇒** A path from vertex 'u' is a sequence of vertices each adjacent to the next.

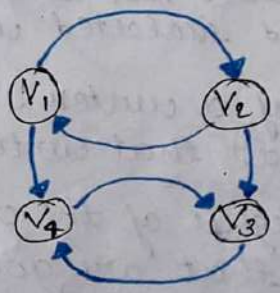
**Cycle ⇒** Cycle is a path in which first and last vertices are the same.

Ex = 1, 2, 3, 1, 1

**Connected Graph ⇒** A graph is called connected graph if there exist a path from to visit any other vertex to any other vertex.

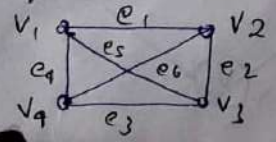



**Directed Graph ⇒** In the directed Graph the direction of the path is given




**Complete Graph ⇒** If an undirected graph of 'n' vertices consist of $\frac{n(n-1)}{2}$ no. of edges then that graph is called a complete graph

Eg :-




Scanned by CamScanner

$n = 4$

$$e = \frac{n(n-1)}{2} \Rightarrow \frac{4(4-1)}{2}$$

$$= \frac{4 \times 3}{2} \Rightarrow 6$$

**Weighted Graph** ⇒ A graph is said to be weighted graph, if edge in the graph is assign some value or weight.

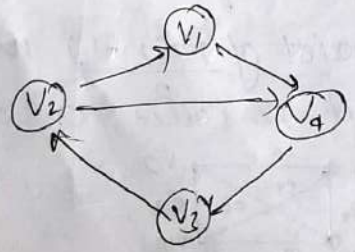
The weight is +ive value that may be representing the cost of moving along the edge or distance b/w the vertices.

**Degree** ⇒ No. of edges incident on a vertex determine it's degree.

The degree of vertex 'u' is written as degree (u), if it is equal to zero that means vertex 'u' does not belong to any edge. This vertex is called isolated vertex.

① In degree ⇒ Indegree of a vertex is the no. of edges that incident to that vertex

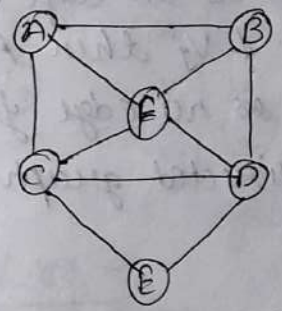
② Out degree ⇒ Out degree of the vertex is total no. of edges that are going away from the vertex.




| vertices | indegree | outdegree |
|---|---|---|
| $V_1$ | 1 | 1 |
| $V_2$ | 1 | 2 |
| $V_3$ | 1 | 1 |
| $V_4$ | 2 | 1 |

**Difference B/w. cycle & Hamiltonion Cycle**

**Hamiltonion Cycle** ⇒ Hamiltonion cycle is also a cycle which having a closed ~~path~~ walk through the graph But the vertices in the cycle are not repeated at all. If other works there is no ~~no~~ repeation of vertex is not allowed.




Hamiltonion Cycle ⇒ A-B-D-E-C-F-A

**Difference B/w Tree & Graph**

| Tree | Graph |
|---|---|
| ① It is collection of nodes & edges. | It is collection of vertices & edges. |
| ② Tree consist of node having any no. of child nodes. But in case of Binary tree every node can have at the most 2 child nodes | ② ~~Ea~~ Each vertex can ~~node~~ have any no. of edges. |

Scanned by CamScanner

| | |
|---|---|
| ② There is a unique node called root in tree. | ③ There is no unique node called root in graph. |
| ④ There will not be any cycle. | ④ A cycle can be formed. Remember ~~there will not be any cycle~~. |
| ⑤ Application → Shortest path or Networking | Application → Decision tree, gaming tree |

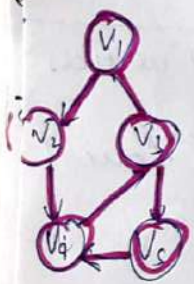
## Graph representation ⇒

① Adjacency matrix ⇒ In this representation a matrix or 2D-Array is used to represent a graph.

The matrix 'A' for a graph G(V, E) with 'n' vertices, if there is an edge represent in B/w vertices $V_i$ & $V_j$ then $A[i][j]=1$ else $A[i][j]=0$ if there is no edge from $V_i$ & $V_j$.

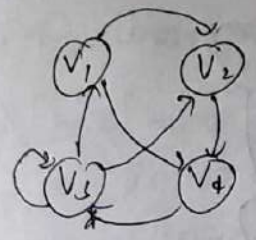
NOTE ⇒ For an undirected graph if $A[i][j]=1$ for $A[j][i]=1$

$$A[i][j] = \begin{cases} 1 & \text{if there is an edge b/w } (V_i, V_j) \\ 0 & \end{cases}$$



$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline V_1 & 0 & 1 & 1 & 0 & 0 \\ V_2\ 2 & 1 & 0 & 0 & 1 & 0 \\ V_3\ 3 & 1 & 0 & 0 & 1 & 1 \\ V_4\ 4 & 0 & 1 & 1 & 0 & 1 \\ V_5 & 0 & 0 & 1 & 1 & 0 \end{array}$$

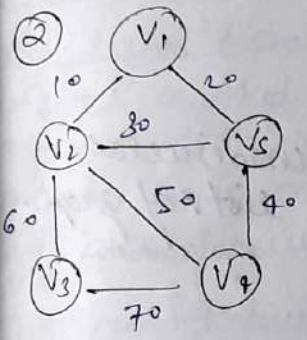
Ex ⇒



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

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

②

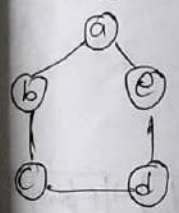


$$\begin{array}{c|ccccc} & V_1 & V_2 & V_3 & V_4 & V_5 \\ \hline V_1 & 0 & 10 & 0 & 0 & 20 \\ V_2 & 10 & 0 & 60 & 50 & 30 \\ V_3 & 0 & 60 & 0 & 70 & 0 \\ V_4 & 0 & 50 & 70 & 0 & 40 \\ V_5 & 20 & 30 & 0 & 40 & 0 \end{array}$$

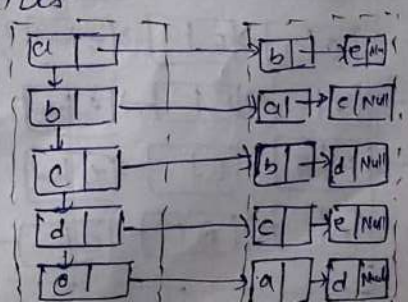
Adjacency matrix representation

## Argency list ⇒

In this representation we store a graph as a link structure. We store all the vertices in a list and then for each vertex we have a link list of its adjacent vertices.

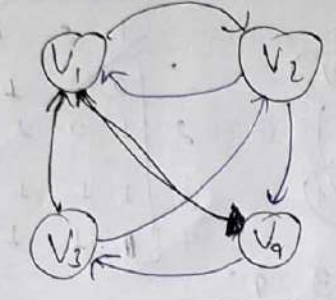


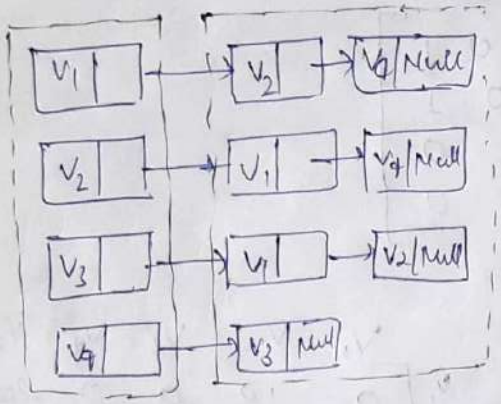
→



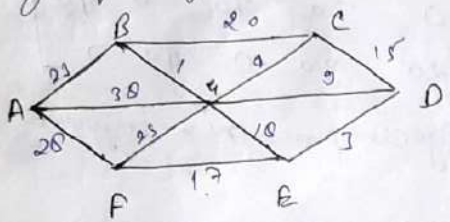

Scanned by CamScanner

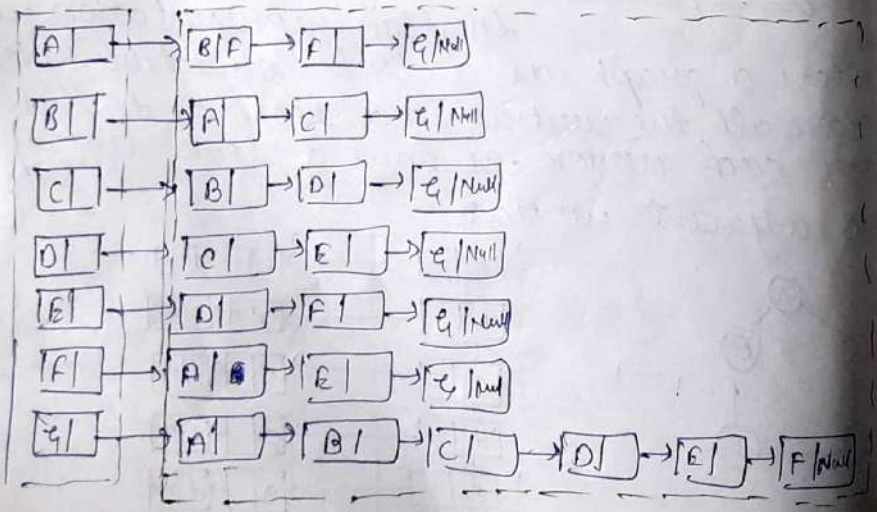
Qu




Qu3 Consider the following undirected graph, find the adjacent list of graph representation




GRAPH Traversal ⇒ A graph traversal means visiting all the nodes of the graph. It may be in many of applications & There may be many methods for visiting the vertices of the graph. There are two graph traversal techniques are used

① BFS (Breath first search)
② DFS (Depth First search)

① Breath first search ⇒ In BFS one node is selected (starting node or root node) is visited marked then all unvisited nodes adjacent to the next node are visited and marked upto so on untill the entire graph has been traversed.

Or

We can traverse a graph level by level. (Horizontly starting from the initial node to the last terminal node.

(Queue data structure is used)

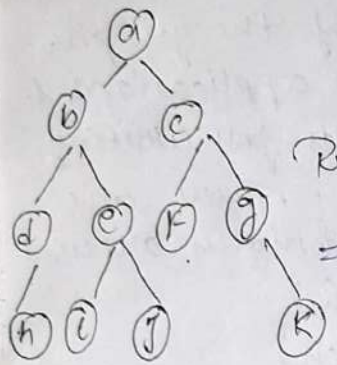
properties of BFS of Binary Tree ⇒

① check All connections at the time.
② Traverse a graph in breadthwise motion.
③ Queue DS is used.
④ BFS works level by level.

Scanned by CamScanner

BFS Graph traversal of BF

Que ~~a b c d e f g h i j k~~




whatever u have to deque pin in result

Result ⇒

⇒ a b c d e f g h i j k

Step 1 Enque 'a'

St 2 Check all connection of 'a' that 'b' & 'c' enque in Quue & dequue 'a'.

Step 3 check all con^n of 'b' that 'd' & 'e' enque in Quue & deque 'b'.

Step 4 check all con^n of 'c' that 'f' & 'g' enque in Quue & deque 'c'.

Step 5 check all con^n of 'd' that 'h' enque in Quue & deque 'd'

Step 6 Check all con^n 'e' that 'i' & 'j' enque in Quue & deque 'e'

Step 7 check all con^n of 'f' that has no con^n & deque 'f'

Step 8 check all con^n of 'g' that ~~has~~ 'k' enque in Quue & deque 'g'

Step 9 check all con^n of 'h' that has no con^n & deque 'h'.

Step 10 check all con^n of 'i' it has no con^n & deque (i)

Step 11 check all con^n of 'j' has no con^n & deque j

Step 12 check all con^n of 'k' has no con^n & deque k'

Result ⇒ a b c d e f g h i j k

Pseudo Code ⇒

```
Enque (root)
while ( ! is empty · quue )
  {
  root = deque ( );
  printf (root);
  if (root → left)
    {
    Enque (root → left)
      {
      if (root → right)
      {
      Enque (root → right);
    }
  }
}
```

Scanned by CamScanner

## DFS ⇒ (Depth first search)

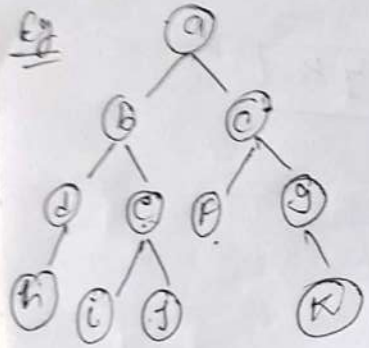
Properties ⇒

① 1 connection at a time

② Stack DS is used < push / pop

③ Similar to preorder Traversal

Eg



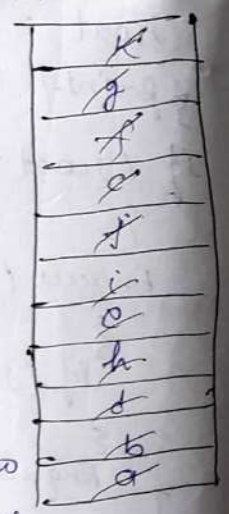
Write Array ⇒

| a | b | d | h | e | i | j | c | f | g | k |
|---|---|---|---|---|---|---|---|---|---|---|

Result ⇒ abd h eijcfgk

Suppose 'a' have two child (left or right), then go left child 'b' & exploring.

Stack



Step 1 'b' i.e going to depth.

Step 2 After exploring 'b' there are 2 conn, we have to check 'd'

Step 3 check conn of 'a' i.e 'c'

Step 4 No possible conn of 'h' so no conn is possible go to stack & pop top element of stack i.e b

Time complexity ⇒ O(n)

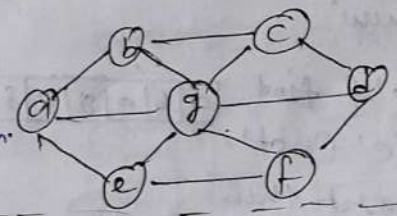
Pseudo code ⇒

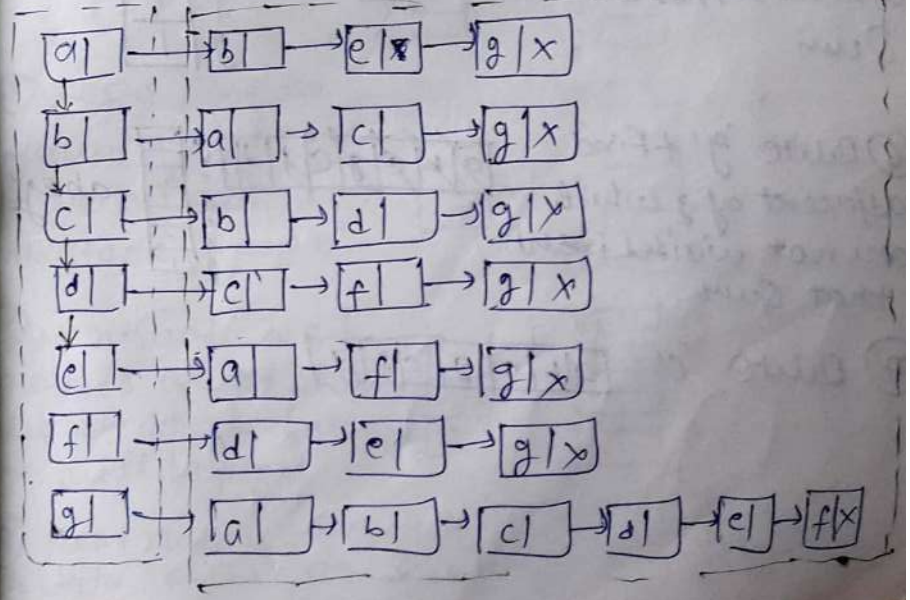
```
DFS (node * root)
  {
    if (root)
    {
      pf (root);
      DFS (root → left);
      DFS (root → right);
    }
  }
```

GRAPH Traversal of given graph ⇒

Find BFS

① We create Adjacency list of given graph.






Scanned by CamScanner

| Action | Queue | visited array | o/p |
|---|---|---|---|
| ① Insert a in Queue & mark visited (a=1) | a | a b c d e f g: 1 | — |
| ② Delete 'a' | | | a |
| ③ find adjacent of a that are b, e, g & insert that in Queue. Marking visited | a b e g | a b c d e f g: 1 1 1 1 1 1 1 | a |
| ④ Delete 'b' find adjacent of 'b' which are not visited, insert them to 'Queue' | a b e g | | ab |
| ⑤ Delete 'e' & find adjacent of 'e' (not visited) insert them Queue | a b e g c f | a b c d e f g: 1 1 1 1 1 1 1 | abe |
| ⑥ Delete 'g' & find adjacent of g which are not visited insert that Queue | a b e g c f d | a b c d e f g: 1 1 1 1 1 1 1 | abeg |
| ⑦ Delete 'c' | a b e g c f d | | |

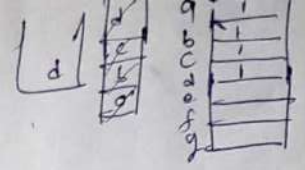
DFS ⇒

DFS

| Action | stack | V.A | o/p |
|---|---|---|---|
| ① Stack with 'a' pop a | a | a b c d e f g: 1 | a |
| ② pop a find the adjacent of b & mark it visited push it onto stack & pop 'b' | b a | a b c d e f g: 1 1 | a,b |
| ③ Next node is c & mark it visited push also the stack. Later 'c' will be popped | c b a | a b c d e f g: 1 1 1 | |
| ④ Next node is 'd' & mark it visited 'push' and stack | | | |

Scanned by CamScanner

lated 'd' will be popped



(5) Next node 'f' & push f on to stack later will be popped.

(6) Next node.

Find out degree of each node for an given Graph & apply BSF algo of to obtain the graph node traversal sequence (traversal start node 1)

Difference b/w BFS & DFS →

| DFS | BFS |
|---|---|
| 1. In DFS we traverse the graph as per depth. | 1. In BFS traversing of graph is in breath vise (level by level manner) |
| 2. Stack DS is used | 2. Queue DS is used for traversing. |
| 3. DFS requires less memory. | 3. BFS requires more memory. |
| 4. DFS can't obtain shortest path. | 4. BFS will find the shortest path if the weight ~~the~~ on links are uniform |
| 5. DFS will perform better in case of large complex graph. | 5. perform poor for large no. of vertices in graph. |

Application of graph ⇒

① Computer networks. (LAN WAN)
② Teliphone cabling graphtheory
③ Job scheduling.

Scanned by CamScanner

# Spanning Tree ⇒

A spanning tree 'S' is a subset of a tree 'T' in which all the vertices of tree 'T' are present but in may not contain all the edges.

## Minimum Spanning Tree (MST) ⇒

(edge select ~~to~~ acc. to the weight)

A minimum spanning tree is a spanning tree which has minimum weight, all the vertices of tree are present in it But it ~~not~~ may not contain all the edges i.e It is the cheapest of edges that keeps the graph in connected mode.

OR.

MST is a subset of edges of a connected of a connected edge weighted undirected graph that connects all vertices together without any cycle, and with the minimum possible edge weight.

Properties ⇒

① possible multiplicity.

② Uniqueness.

③ Minimum Cost edges

④ Contradiction. (विरोध, खंडन)

⑤ Cycle property.

Applications ⇒ Computer Networks, Tele communication networks, electric grid. ~~Transfer~~ Transportation, clustering, image registration, circuit design, taxnomy etc.

⇒ Now we will discuss two popular algo of MST.

1) Prim's Algorithm.

2) Krushkal's algorithm.

① Prim's Algorithm ⇒ It is a MST algo for a weighted undirected graph, that means it finds a subset of edges that form a tree & include every vertex where the total weight of edges is minimized.

Algo.

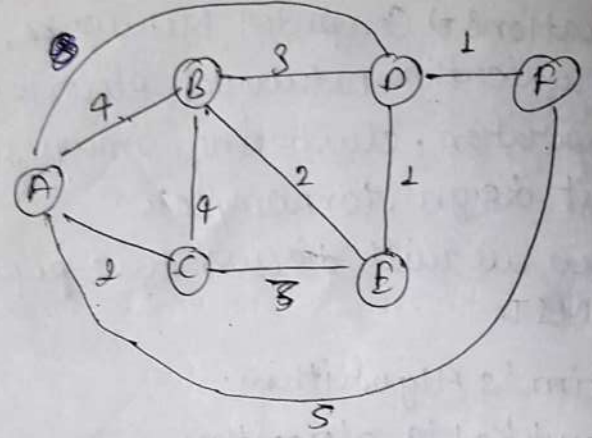
Step 1 Select the pair with minimum weight

S2 Select the adjacent vertex of minimum weight & select the min weighted edge using this adjacent vertex. The selected adjacent vertex should not form any cycle.

S3 Repeat step 1 & 2 ~~ch~~ until all the vertices are getting covered.

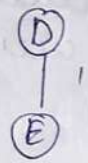
Eg ⇒

Scanned by CamScanner

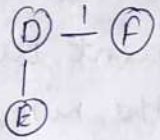



Soln which vertex has edge minimum weight.

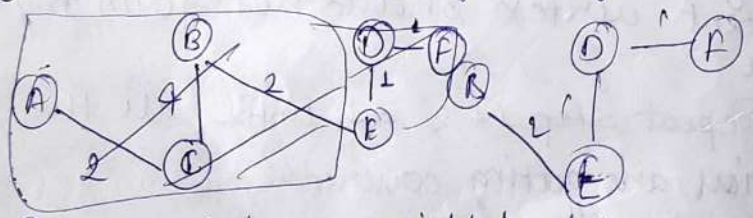
① Select minimum weighted ~~graph~~ edge from given graph



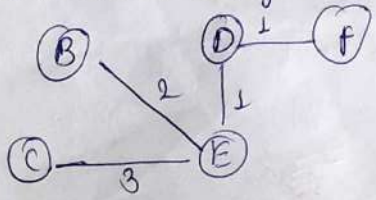

② Select minimum weighted ~~edge~~ -



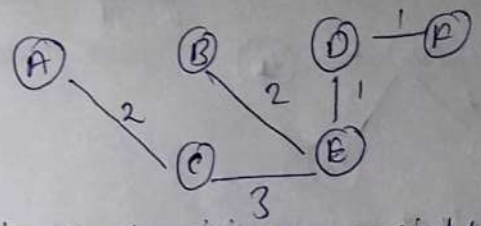

③ Select minimum weighted edge -




④ Select minimum weighted edge -




⑤ Again next minimum weighted edge.



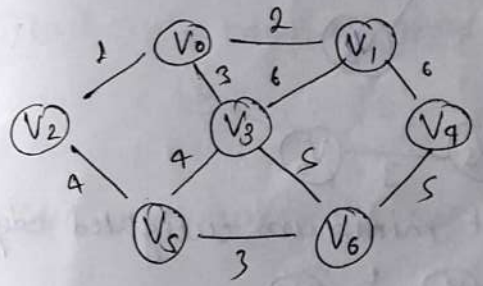

⑥ Again next minimum weighted edge ~~there~~ Thus there is no edge which contain minimum edge. ~~so~~ ~~can~~ If we select minimum edge it will contain a cycle.

So minimum length of the graph is

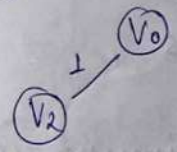
length = 2 + 2 + 3 + 1 + 1

= 9

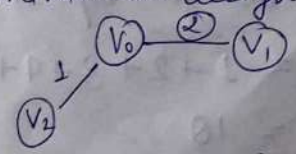
②




① Select minimum weighted edge from the given grap.

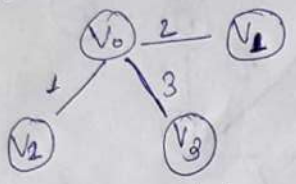



② Select minimum weighted graph.

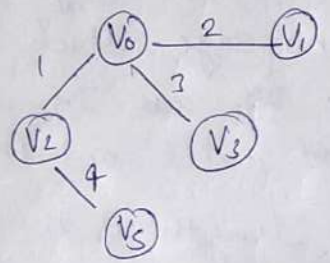



Scanned by CamScanner

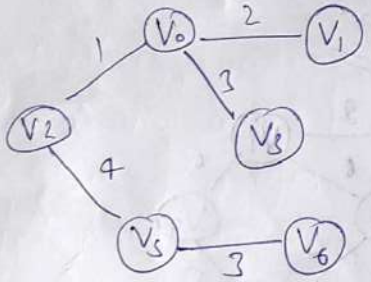
③ Again select minimum ~~graph~~ weighted edge.



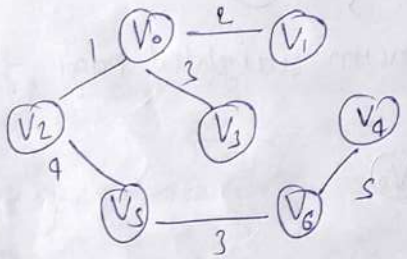

④ Again select minimum weighted edge




⑤ Again select minimum weighted edge.




⑥ Again select minimum weighted edge.

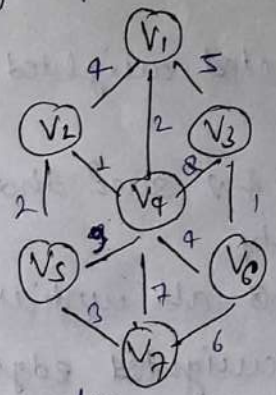



⑦ Now, There is no minimum weighted edge so minimum length of the graph.

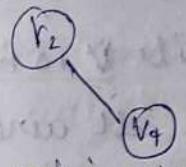
Minimum length = 1+2+3+4+3+5

= 10

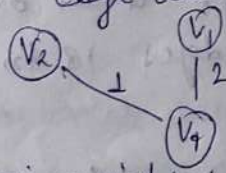
③ Find the minimum spining tree using prism graph.



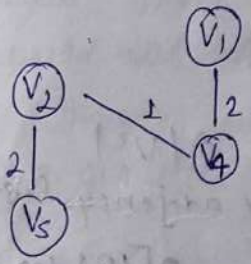

① Select min. edge -



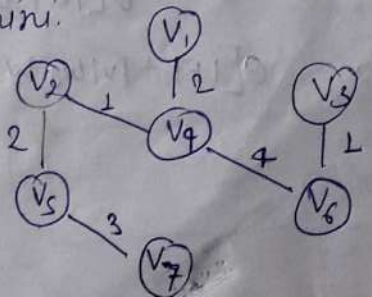

② Select mini. edge weighted. -




③ Select mini weighted edge -




④ Select mini.




= 13

Scanned by CamScanner

## Prim's Algo

$V = (V_0, V)$

**Algo**

Let $(V, E)$ is connected weighted undirected graph.

S1 Create 2 sets $V$ & $V'$ such that
- $V'$ is empty
- $V$ contains all vertices of graph.

S2 Select min weighted edge $(i, j)$ from $V$ & insert $i$ & $j$ in set $V'$

S3 Repeat steps (4) while $V$ is not equal to $V'$

S4 Find all neighbours of all vertices which are in set $V'$ such that one end point of neighbouring edges is in $V'$ & another not in $V'$

S5 Sum up all selected edges weight (S)

S6 Exit

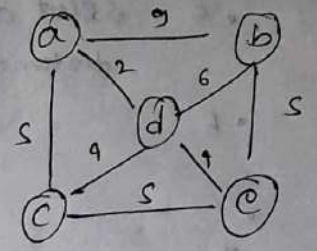
Time complexity $\Rightarrow$ $O(|V|^2)$

Binary ~~heap complexity~~ adjency list -

$O[|E| \log |V|]$

$O[|E| + |V| \log |V|]$ = Fibbnaci heap.

**Qu**




## Kruskal's algorithm

It is MST algo, which find an edge of least possible weight that connects any two trees in the forest.

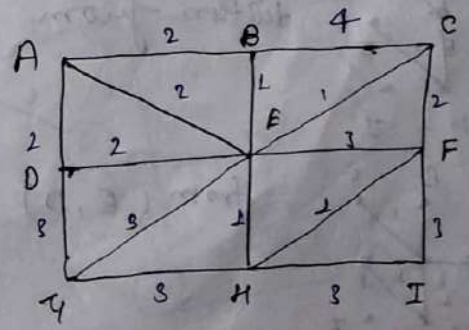
It finds subset of edges that forms a tree that include every vertex, where the total weight of all edges in the graph is minimize

**Algo**

S1 Select the pair with minimum weight each time and make the union of the selected edges. The selected adjacent vertex should not form any cycle or ckt.

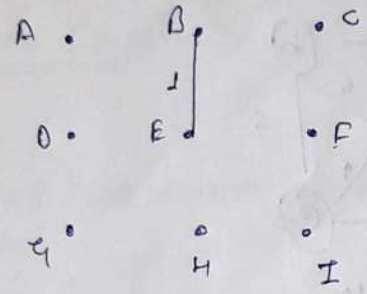
S2 Repeat step 1 untill all the vertices are getting covered.

**Qu**



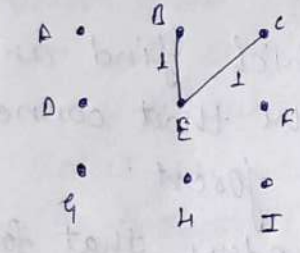

Scanned by CamScanner

S1



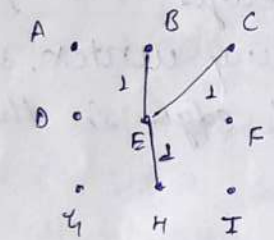
select an edge with min. distance from (B, E)

S-2



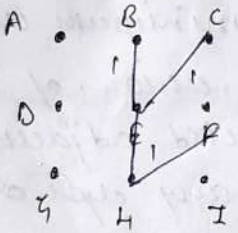
Again an edge with distance from (E, C)

S3



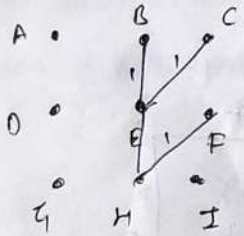
Again select edge with min. distance from (E, H)

S-4



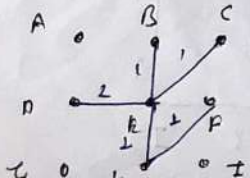
Again select edge with min. distance from (E, F)

S-5



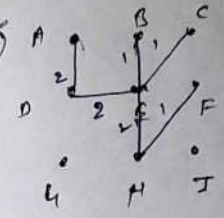
Again take min. edge with distan from

S-6



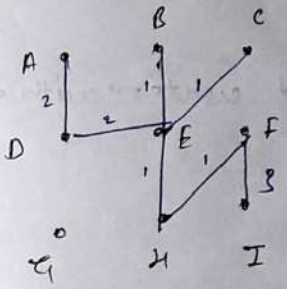
from (E, D)

S-7



from (D, A)

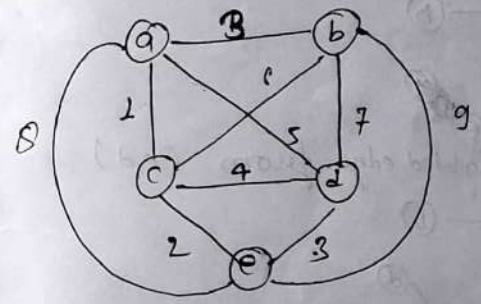
S-8

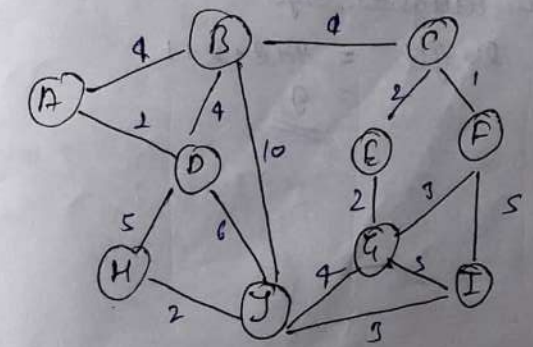


Total Distance ⇒ 2 + 2 + 1 + 1 + 1 + 1

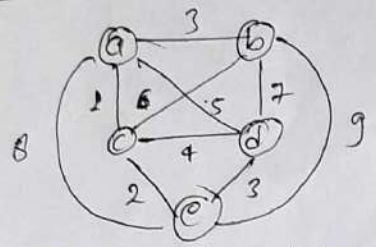
⇒

Q Construct the spining tree of given graph using prism's & Kruskal.

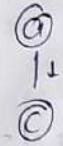




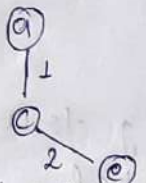
① By Prism's Algo →



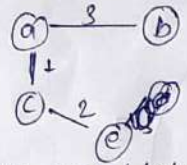

S1 We select mini weighted vertex with edge from (a, c)



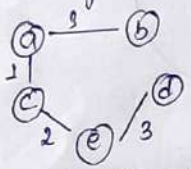

S2 Again we select mini weighted edge from (c, e)




S3 Again select mini. wt. edge from (a, b)




S4 Select mini. weighted edge from (e, d)

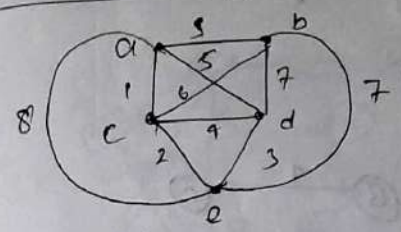



S5 ~~Select mini weighted edge~~

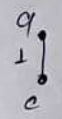
So total length = 8+3+2+1

= 9

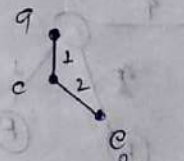
(ii) By Kruskal's Algo →



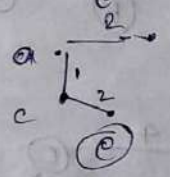

S1 from (a, c)



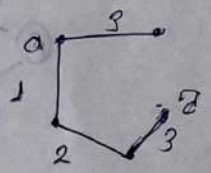

S2 from (c, e)




S3 from (a, b)




S4 Select weighted edge -

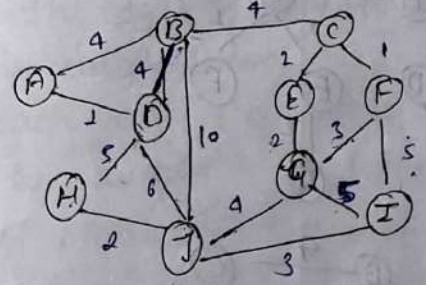



Step5 Select mini weighted graph.

So total distance = 8+3+2+1

= 9.

②

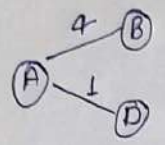



By Prism's Algo →

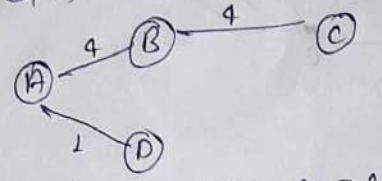
① Select mini. weighted edge -

Scanned by CamScanner

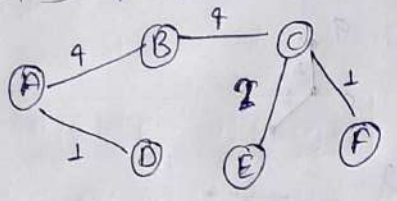
S-2 from (A, B)



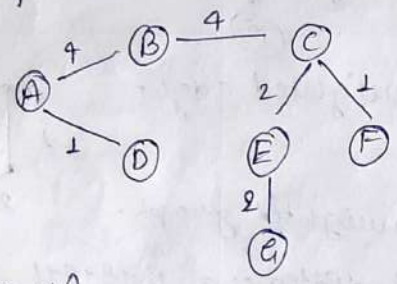

S-3 from (B, C)



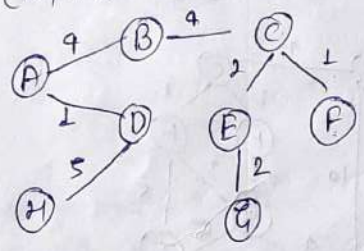

S-4 from (C, F) & from (C, E)



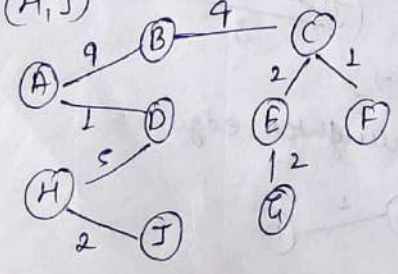

S-5 from (E, G)



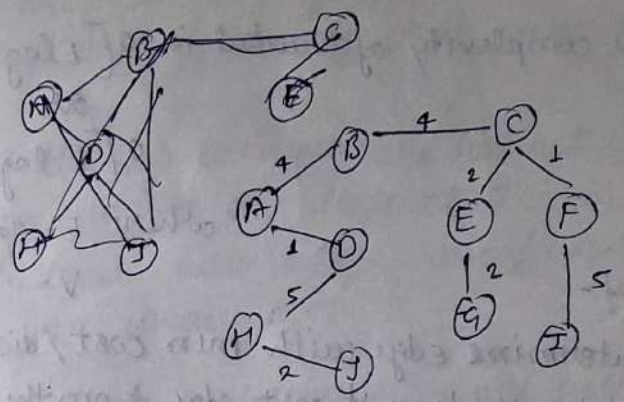

S-6 from (D, H)




S-7 from (H, J)




S-8 from (F, I)




S-9 Total length of MST = 1+4+4+5+2+2+1+2+5

= 26

## Algo. of Kruskal's =>

MST Kruskal's (G, w)

(i) $A \leftarrow \phi$ {is empty set}

(ii) for each vertex $v \in V(G)$

(iii) do - make set (v)

(iv) Sort the edges of E into non decreasing order by weight 'w'.

(v) for each edge $(u, v) \in E$, taken in nondecreasing order by weight.

(vi) do if FIND-SET(u) $\neq$ FIND-SET(v)

(vii) Then $A \leftarrow A \cup \{(u, v)\}$

(viii) UNION (u, v)

(ix) return A

Scanned by CamScanner

## Complexity Analysis ⇒

Time complexity of Kruskal is $O[E \log V]$

or

$O[E \log E]$

where E = Total no. of edges

V = " " " vertices

proof:-

(i) Determine edge with min cost / distance

(ii) Delete the lowest cost edge from the list of edges.

$$\boxed{O[E \log E] = O[E \log V]}$$

### Difference b/w prism & Kruskal's

| prism's | Kruskal's |
|---|---|
| (i) This algo is for obtaining MST by selecting the adjacent vertices of already selected vertices. | (i) This algo is for obtaining MST but it is not necessary to choose adjacent vertices of already selected vertices |
| (ii) prism algo. the tree/graph that we are making is always remains connected. | (ii) In this algo the tree/graph we are making is always remains disconnected |
| (iii) This algo. intilizes with node or vertex | Intilizes with edge. |
| (iv) Prism algo is faster for dense graph. | (iv) faster for sparse graph |
| (v) complexity of prim's algo is $O[V^2]$ | (v) Complexity of Kruskal's algo is $O[E \log E]$. |
| (vi) we prefer this algo when there are large no. of graph like $E = O(V^2)$ | (vi) we prefer this algo when there are less no. of graph like $E = O(V)$ |

NOTE ⇒ Will the prism and Kruskal produce the same MST in any graph?

Ans: In this case following two case are possible -

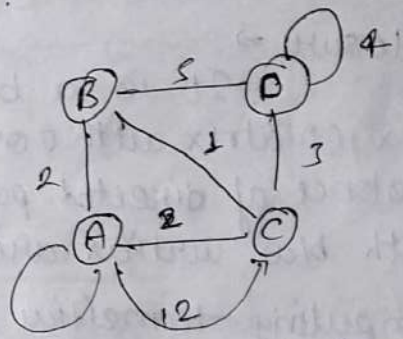
① When all the edge weights are distinct.

If all edge weights are distinct It is garnteed that both algo are find the same i.e you unique MST.

② When all the edge weights are not distinct

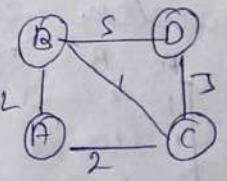
If all edge weights are not distinct then both algo's may not always produce same i.e unique MST, But the cost of MST would always be same in both the cases.

(8)

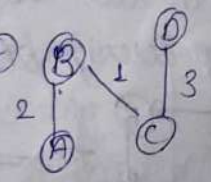



① we have to remove { self loop, parallel edge }

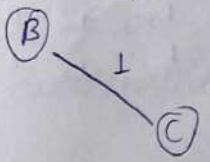
(2)




(4)



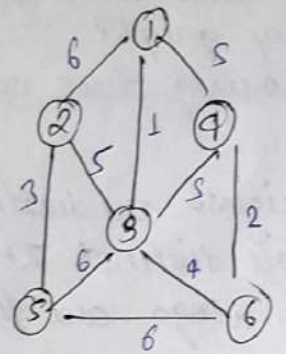

(3)




= 6

Scanned by CamScanner

Que



| Edges | Cost / distance |
|---|---|
| 1-3 | 1 |
| 4-6 | 2 |
| 2-5 | 3 |
| 3-6 | 4 |
| 1-4 | 5 |
| 2-3 | 5 |
| 3-4 | 5 |
| 3-5 | 6 |
| 5-6 | 6 |
| 1-2 | 6 |

Edges ⇒ {1+2+3+4+5} = 15

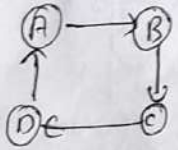
Vertex ⇒ {1, 3, 4, 6, 2}

Transitive Closure ⇒

It is a basically Boolean matrix (Matrix with 0 or 1 value) in which existence of directed path of arbitrary length b/w vertices are mention.

While computing transitive closure we have to start with some vertex and have to find all the edges which are reachable to every other vertex.

Eg :-



T.C ⇒

| | A | B | C | D |
|---|---|---|---|---|
| A | 1 | 1 | 1 | 1 |
| B | 1 | 1 | 1 | 1 |
| C | 1 | 1 | 1 | 1 |
| D | 1 | 1 | 1 | 1 |

The reachable edges for all the vertices to be obtain.

Shortest path problem ⇒ It is a classic problem in a graph which is useful to find shortest path / distance between any two nodes (Source to Destination) in a weighted graph.

Types of Shortest path

- Single source shortest path (SSSP)
  - Cost / distance of S.P from source vertex to destination vertex (V)
  - (Dijkstra's Algo)
- All pair shortest path (APSP)
  - cost of S.P from each vertex to every other vertex
  - Floyd Warshall's Algo

Dijkstra's Algo

It is used to find shortest path between source node to destination node for a weighted graph. It's named after its discover Dutch Computer Scientist. It maintain a set 'S' of vertices whose final shortest path weights, from the source 's' having been determine.

Scanned by CamScanner

## Algo of Dijkstra

① Initialize Single Source (G, S)

S-1) $S = \phi$ , $S = 0$

S-2 $Q$ = All the vertex 'V' of 'G'

S-4 while ($Q \neq 0$)

$u$ = Extract - min (Q)

$S = S \cup \{u\}$ or $S + \{u\}$

for each vertex $v \in G$ ; Adj [v]

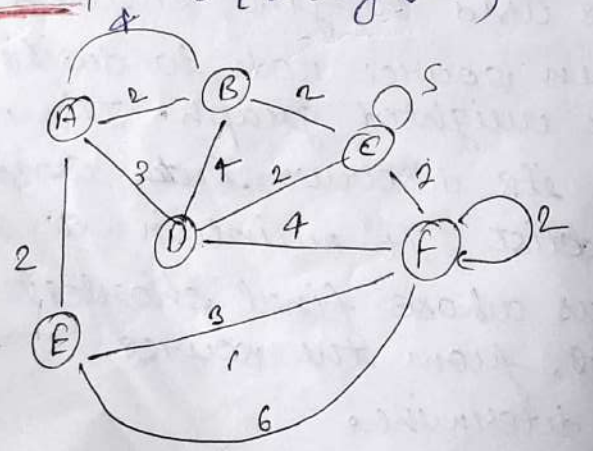
Relax Func (u, v, w) ← edges

if [dis [v] > dis [v] + length (u, v)]

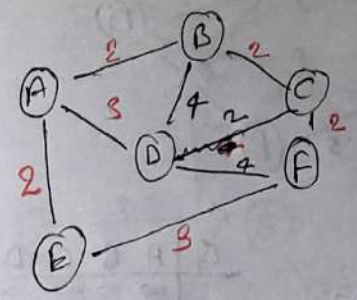
return S

Time Complexity $O[E \log |V|]$

Que




Soln S-1 Remove self loop & parallel edgs.




New distn value = min [old distn value, marked value + edge weight]

S-2 Creat weight matrix.

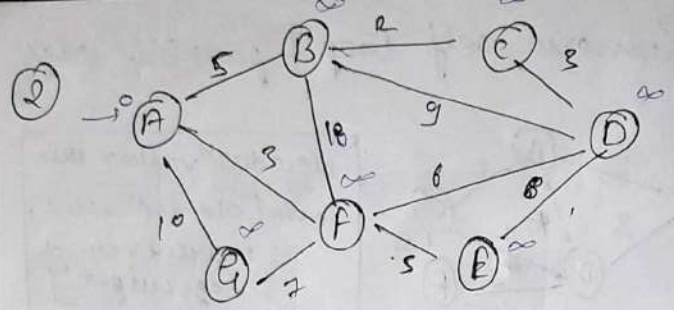
| Marked or Selected vertex | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | [0] | ∞ | ∞ | ∞ | ∞ | ∞ |
| B | | [2] | ∞ | 8 | 2 | ∞ |
| E | | | 4 | 8 | [2] | ∞ |
| D | | | 4 | [3] | | 5 |
| C | | | [4] | | | 5 |
| F | | | | | | [4] |

① Set '0' to Source vertex & '∞' to remaining vertex.

② Select the smallest unmarked value & mark that vertex

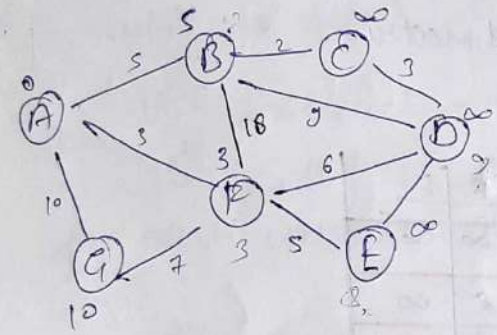
Scanned by CamScanner

(2)

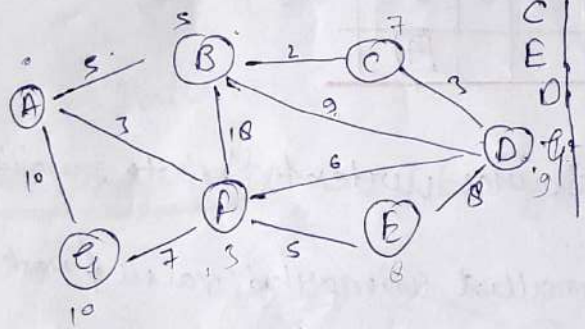


$S = \{A\}$

| Q | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| A | 0 | 5 | ∞ | ∞ | ∞ | ∞ | ∞ |

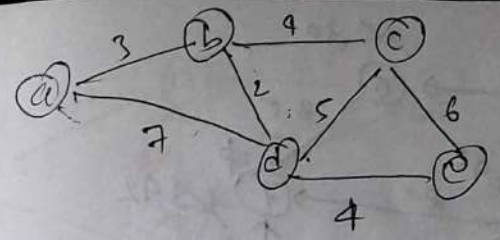


| Q | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| A | 0 | 5 | ∞ | ∞ | ∞ | $\boxed{3}$ | 10 |
| F | | $\boxed{5}$ | ∞ | 9 | 8 | | 10 |
| B | | | $\boxed{7}$ | 9 | 8 | | 10 |
| C | | | | 9 | $\boxed{8}$ | | 10 |
| E | | | | $\boxed{9}$ | | | 10 |
| D | | | | | | | 10 |
| G | | | | | | | |

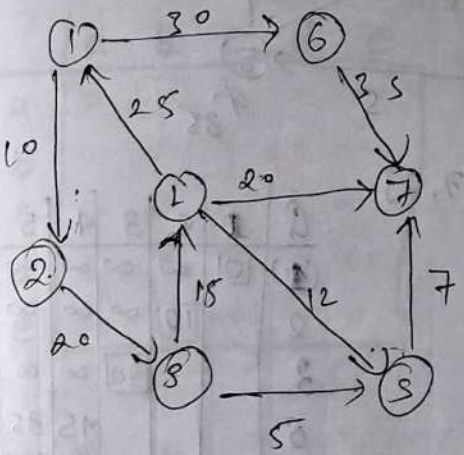


$S = \{A, F, B, C, E, D, G\}$

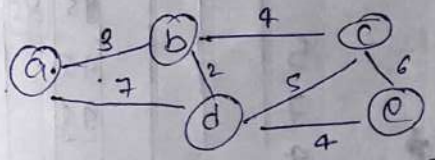
(2)

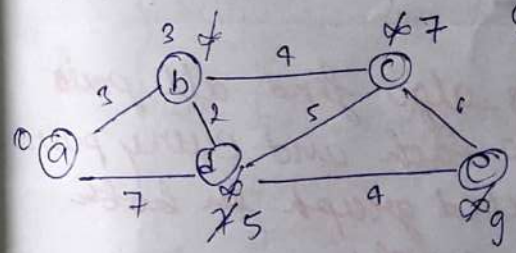


(2)



(2)

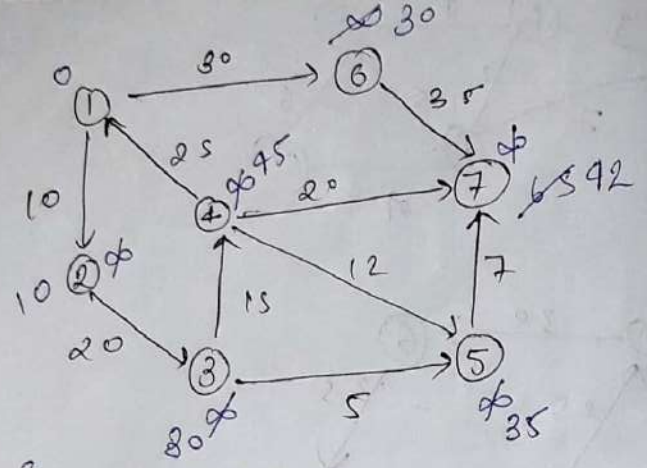




| Q | a | b | c | d | e |
|---|---|---|---|---|---|
| | $\boxed{0}$ | ∞ | ∞ | ∞ | ∞ |
| a | | $\boxed{3}$ | ∞ | 7 | ∞ |
| b | | | 7 | $\boxed{5}$ | ∞ |
| d | | | $\boxed{7}$ | | 9 |
| c | | | | | $\boxed{9}$ |
| e | | | | | |

$S = \{a, b, d, c, e\}$

Scanned by CamScanner




$S = \{1, 2, 3, 6, 5, 7, 4\}$

| S | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | | 10 | ∞ | ∞ | ∞ | 80 | ∞ |
| 3 | | | 80 | ∞ | ∞ | 80 | ∞ |
| 6 | | | | 45 | 85 | 80 | ∞ |
| 5 | | | | 45 | 85 | | 65 |
| 7 | | | | 45 | | | 42 |
| 4 | | | | 45 | | | |

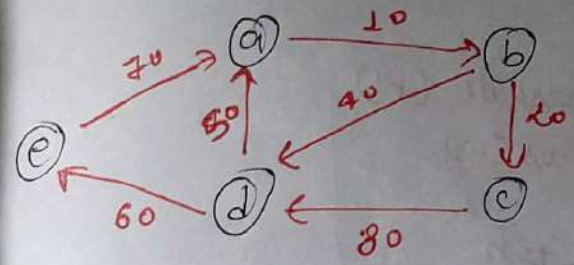
## Floyd warshall :

~~Flyd warsal :~~

This algo find all pair shortest path b/w each and every pair of vertices in both directed graph ~~in both~~ and undirected graph.

This algo was invented by Robert Floyd & Stephn Warshall.

In floyd warshall positive and negative edges (weights) are allowed.




⇒

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 10 | ∞ | ∞ | ∞ |
| b | ∞ | 0 | 20 | 40 | ∞ |
| c | ∞ | ∞ | 0 | 30 | ∞ |
| d | 50 | ∞ | ∞ | 0 | 60 |
| e | 70 | ∞ | ∞ | ∞ | 0 |

K → intermidiate nodes

$d_{ij}$ = wt of S.P from vertex 'i' to 'j' from which all intermidiate node (vertex) are in $(1, 2, \ldots k)$

$$d_{ij}^{K} = \begin{cases} w_{ij}, & \text{if } K = 0 \\ \min d_{ij}^{(K-1)}, \; d_{iK}^{(K-1)} + d_{Kj}^{(K-1)} & \text{if } K \geq 1 \end{cases}$$

previous node

For $V \times V$ Matrix – ($4 \times 4$)

$$D^{(m)} \Rightarrow d_{ij}^{(m)} \;\&\; d_{ij}^{(n)} = \begin{cases} 0, & \text{if } i = j \\ w_{ij} & \text{if } i \neq j \\ \infty & \text{otherwise} \end{cases}$$

Scanned by CamScanner

Algo.

Floyd Warshall (W)

(n) = socus (w)

$D^0 \to w$

for $k \to 1$ to n

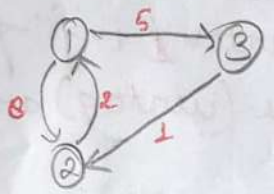
do for $i \to 1$ to $n \to \{i,j \to$ Combined value of vertices$\}$

$j \to 1$ to n

do $d_{ij}^{k} \to \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

return $(D^n)$

Q




"* with respect to source all diagonal value of matrix always be zero"

Step 1

K=0

$D^{(0)}$ =

(if i=j)

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 8 | $\boxed{5}$ |
| 2 | 2 | 0 | ∞ |
| 3 | ∞ | 1 | 0 |

Step 2

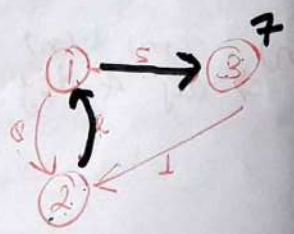
K=1

$D^{(1)}$ =

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 8 | 5 |
| 2 | 2 | 0 | 7 |
| 3 | ∞ | 1 | 0 |

path * 2-3 ⇒

1+2 = 7 < ∞




Step 3

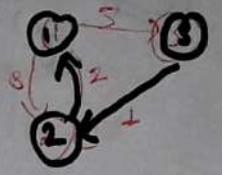
(K=2)

$D^{(2)}$ =

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 8 | 5 |
| 2 | 2 | 0 | 7 |
| 3 | 3 | 1 | 0 |

path (3-2-1) =

1+2+1 < ∞




Step 4

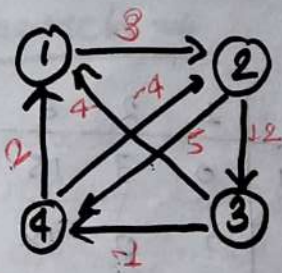
$D^3$ is computed because '1' b/w '3'

(K=3)

$D^{(3)}$ =

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 6 | 5 |
| 2 | 2 | 0 | 7 |
| 3 | 3 | 1 | 0 |

Que




K=0

$D_{ij}^{(0)}$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | $\boxed{3}$ | ∞ | ∞ |
| 2 | ∞ | 0 | 12 | 5 |
| 3 | 4 | ∞ | 0 | -1 |
| 4 | 2 | -4 | ∞ | 0 |

Step 2

K=1

$D_{ij}^{(1)}$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 15 | 8 |
| 2 | ∞ | 0 | 12 | 5 |
| 3 | 4 | ∞ | 0 | -1 |
| 4 | 2 | -4 | ∞ | 0 |

Scanned by CamScanner

Step-3

$D^{(2)}$ K=2 =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 15 | 8 |
| 2 | 7 | 0 | 12 | 5 |
| 3 | 4 | ∞ | 0 | -1 |
| 4 | 2 | -4 | ∞ | 0 |

Step-4

$D^{(3)}$ K=3 =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 15 | 8 |
| 2 | 7 | 0 | 12 | 5 |
| 3 | 4 | -5 | 0 | -1 |
| 4 | 2 | -4 | ∞ | 0 |

Step 5

$D^{(4)}$ K=4 =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 3 | 15 | 8 |
| 2 | 7 | 0 | 12 | 5 |
| 3 | 4 | -5 | 0 | -1 |
| 4 | 2 | -4 | 8 | 0 |

# Searching & Sorting

Searching ⇒ Searching is techinque from of finding desired element from the given list of element.

Searching Tech.

- Linear Search or Sequential search.
- Binary Search

## Linear Search ⇒

It is a technique in which there is no specific ordering of data. The key element i.e (the element which to be search) is compare against every element in the list from beginning to the end of the list.

Linear Search is use to search an order list of elements (array in which data element are not searched.

Scanned by CamScanner

Array

| | RollNo. | Name | Marks |
|---|---|---|---|
| 0 | 15 | Kavita | 100 |
| 1 | 2 | Anjali | 90 |
| 2 | 13 | Vandna | 81 |
| 3 | 1 | Aman | 50 |
| 4 | 12 | Moon | 78 |
| 5 | 3 | Surabh | 94 |

① find the student's record which roll-no. is '12' search.

Key roll no = 12

## Algo. Linear Search :

Step:1 L.S (A, N, VAL)

(Intialize) Set Post = -1

Step:2 Set I = 1

Step:3 Repeat Step (iv) I ≤ iv

Step:4 If A[I] = VAL

Set POS = -1

PRINT pos

Go to Step (vi)

Step:5 If pos = -1

PRINT 'VALUE' is not Present in Array.

I ≤ N Step II Exit

## Advantage

① It is not required specific ordering before implying the method

① It is linear

## dis Advantage

Time consuming. Less efficient.

## complexity :

O(n)

## Binary Search :

Binary search is an extremly efficient algo. This search technique finds the given item or an element in minimum possible comparisons. To do the Binary search first we have to short the array elements in following manner

(i) Find middle element of the array.

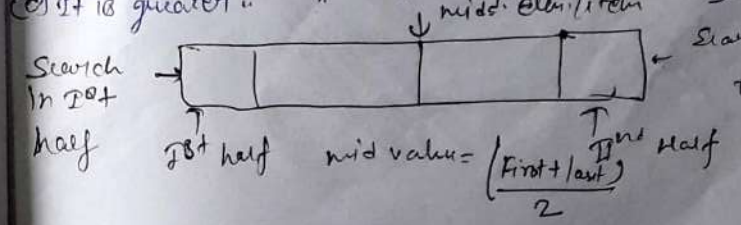
(ii) Compare middle element with an item/element.

(iii) There are 3 case -

(a) If is ~~less than~~ desired element then search is successful

(b) If is less than desired element then search only I^st half the array.

(c) If is greater " " " mid. elem./item " IInd " "




Scanned by CamScanner