20/1/17

Data Structure

A data structure is a particular way of storing and organising data in a computer so that it can be used efficiently. The term data structure refers to a scheme for organising related pieces of information.
The organised collection of data in a mathematical or logical way is called data structure.
The study of data structure includes :-

1. Logical description of data
2. Implementation details of the data structure.
3. Quantitative Analysis of the data structure includes amount of memory required to store the data structure and the time required to process.

Types of data structures

There are mainly two types of data structure :-

1. Linear → Array, linked list, stack, queue.
2. Non-Linear → graph, tree.

Linear Data Structure

A data structure whose elements forms a proper structure sequence, or having a sequential structure or arranged in a linear fashion, every element in the structure has a unique predecessor and unique successor. Examples are array, linked list, stack and queue.

Non-Linear Data Structure

A data structure whose data or elements do not form a sequence, or having a complex structure, data are stored in random manner, having no unique predecessor or successor. For example trees and graph.

Data type

A data type defines a domain of allowed values and the operations that can be performed on those values

```
char a, b, c;
c = a + b;
```

Output

c = (ab) — concatenation

Operation of Data Structure

1. Insertion
2. Deletion
3. Searching
4. Traversing
5. Sorting
6. Merging.

Array

An array is a collection of similar data items.

Abstract Data Type for Array

1. Stores a given no. of elements with a given data type.
2. Write/ modify element at any given specific position.
3. Read elements at a specific position.

Implementation

Array can perform all the operations and we have to use array for this abstract data type.

A → Array
N → No. of elements
K → position
Element → the data which we want to insert at K.

Algorithm for inserting an element into a linear array.

Insert (A, N, K, element). where K <= N.

[Initialise] set J = N.

Repeat step 3 and 4 while (J >= K).

Move the Jth element downwards.

A[J+1] = A[J].

[Decreament] J = J - 1.

[end of loop].

[Store the element] A[K] = element.

set N = N+1.

Exit.

Algo for Deletion of an array.

Delete (A, N, K, element). where K <= N.

[Initialise] set element = a[K]. and set J = K.

~~Repeat for J = K to N-1~~

Repeat step 3 and 4.

while (J <= N-1)

A[J] = A[J+1].

J = J+1.

[end of loop]

Set N = N-1.

Exit.

Homework

Q. Write programs in C++ for the above algorithms.

// Program for insertion of an element in 1-D Array

```
# include <iostream.h>
# include <conio.h>
  void main ( )
  { int A[10], j, element, n, k;
    clrscr ( );
    cout<<"\n enter no. of elements of array\n";
    cin>>n;
    cout<<"\n enter elements of array\n";
    for (j=0 ; j<n ; j++ )
    { cin>> A[j];
    }
    cout <<"\n enter the element to be inserted \n";
    cin>> element;
    cout<<"\n enter the position at which you want to insert the
          element \n";
    cin >> k;
    for(j= n-1; j>= k-1; j--)
    { A[j+1] = A[j];
    }
    A[k-1]= element;
     n++;
    cout <<"\n final array";
    for(j=0 ; j<n ; j++)
    { cout<<"\n" << A[j];
    }
    getch( );
  }
```

// Program for deletion of an element from array.

```
# include <iostream.h>
# include <conio.h>
  void main ( )
```

```
{int A[10], j, element, n, k;
clrscr();
cout << "\n enter no. of elements of array \n";
cin >> n;
cout << "\n enter elements of array \n";
for(j=0; j<n; j++)
{ cin >> A[j];
}
cout << "\n enter the element to be deleted \n";
cin >> element;
cout << "\n enter the position from which you want to delete the element
      \n";
cin >> k;
for(j = k-1; j <= n-1; j++)
{ A[j] = A[j+1];
}
n--;
cout << "\n final array";
for(j = 0; j < n; j++)
{ cout << "\n" << A[j];
}
getch();
}
```

Algorithm for traversing of array.

Insert(A, S, ~~j~~)

1. ~~Repeat steps 2 and 3:~~ [Initialise] I = 0
2. Repeat steps 3 and 4.
3. Add array's element one by one to S. until (j < 10).
   S ← S + A[j].
4. [increament] j = j + 1.
   [end of loop].
5. Exit.

Stack

Stack is a linear data structure and very much useful in various applications of computer science.
A list with restriction that insertion and deletion can be performed only from one end (i.e. top of the stack).
A stack is an ordered collection of homogeneous data elements where the insertion and deletion operations take place at only one end.

Representation of stack

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

1. Using one dimensional array.
2. Single linked list.

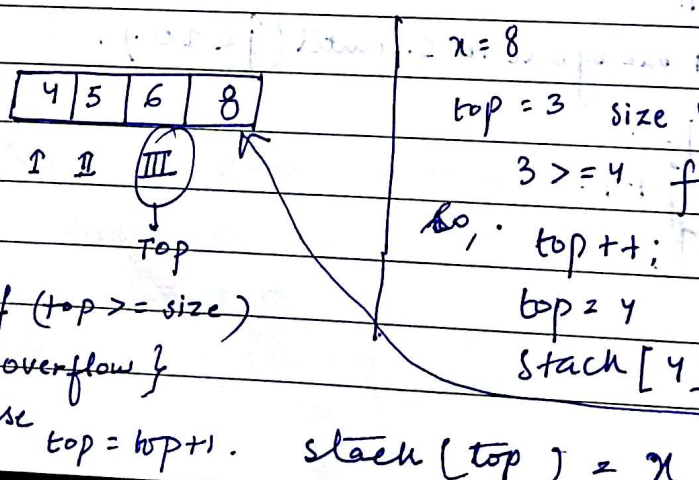
Operations on stack

PUSH (x)
POP (x)
TOP (x)
isempty (x)

Algorithm for PUSH operation

Push (x, size, TOP)

1. If TOP is greater than equal to size then print stack is full. else.
2. TOP is assigned as TOP + 1.
   Stack [TOP] = x
3. Exit



x = 8
top = 3 size 4
3 >= 4 false
so, top++;
top = 4
Stack [4] = 8.

if (top >= size)
{stack overflow}
else
top = top + 1. Stack [top] = x

Algorithm for POP operation

POP (size, top)

1. if (top < 1) then print stack is empty. else.
2. item = stack [top] .
   top = top - 1 .
   [end of ~~loop~~ if]
3. Exit.

Algorithm for TOP operation

Application of Stacks

1. Function calls / recursion.
2. To implement undo operation in an editor.
3. To ~~[illegible]~~ check for balanced paranthesis.
4. Evaluation of postfix expression.
5. Conversion of infix to postfix.

Q: Convert the infix expression into prefix and postfix notation.

((A+B) * (C/D) - E ^ (F*G))

1. Prefix operator operand 1 operand 2
2. Postfix operand1 operand2 operator

Prefix ((+ A B) * (/ C D) - E ^ (* F G))

(* + A B / C D - ^ E * F G)

- * + AB / CD ^ E * FG

((A+B) * (C/D) - E^(F*G))

Postfix

((AB+) * (CD/) - E^(FG*))

AB+CD/* - EFG*^

AB+CD/* EFG*^ -

2. ((A+((B^C)-D)) * (E-(A/C)))

Prefix

~~((A+((^BC)-D)) * (E-(/AC)))~~

~~((A+(-^BCD) * (-E/AC))~~

~~(A+*(-^BCD-E/AC))~~

~~+A*-^BCD-E/AC~~

~~Postfix~~

Prefix

((A+((^BC)-D)) *(E-(/AC)))

((A+(-^BCD)) * (-E/AC))

((+A-^BCD) *(-E/AC))

* + A -^BCD - E/AC

Postfix

((A+((BC^)-D)) * (E-(AC/)))

((A+(BC^D-)) * (EAC/-))

((ABC^D-+) * (EAC/-))

ABC^D-+EAC/- *

(1) (5*4-3+2)/(3*2)

(2) (((6^3)^4-5)+6)

(1) Prefix

(5 * 4 - 3 + 2) / (3 * 2)

(*54 - +32) / (*32)

- *54+32 / *32

/ - *54+32 *32

Postfix

(5 * 4 - 3 + 2) / (3 * 2)

(54* - 32+) / (32*)

54*32+ - / 32*

54*32+ - 32* /

(2) Prefix

(((6^3)^4 - 5) + 6)

((^63^4 - 5) + 6)

((^^634 - 5) + 6)

(-^^6345 + 6)

+ -^^6345 6

Postfix

(((6^3)^4 - 5) + 6)

((63^^4 - 5) + 6)

((63^4^ - 5) + 6)

(63^4^5 - + 6)

63^4^5 - 6 +

((5↑2 - 2) + (6 * 3↑(4 - 3) + 5))

Postfix

((52↑ - 2) + (6 * 3 + (43 -) + 5))

(52↑2 - + (63* + 43 - + 5))

(52↑2 -) + (63*43 - + + 5)

(52↑2 -) + (63*43 - + 5 +)

52↑2 - 63*43 - + 5 + +

(A+(B*C)/(D-(F+(G+H))))

Postfix. (A+(BC*)/(D-(F+GH+)))

(A+(BC*)/(D-FGH++))

(A+(BC*)/(DFGH++-))

(A+BC*DFGH++-/)

ABC*DFGH++-/*+

Algorithm for transformation of infix expression to postfix expression (polish notation).

Polish (Q, P) where Q is an infix expression and P is the postfix expression.

1. PUSH '(' onto the stack and add ')' at the end of Q (infix expression).
2. SCAN Q from left to right and repeat steps 3 to 6 for each element of Q until stack is empty.
3. If an operand is encountered, add it to P.
4. If a '(' is encountered, PUSH it on to the stack.
5. If an operator is encountered then
   (i) Repeatedly POP from stack and add to P each operator which has the same precidence as or higher precidence than operator.
   (ii) Add operator to stack. [end of if structure]
6. If a ')' then.
   (i) Repeatedly POP from stack and add to P each operator until a left paranthesis is encountered.
   (ii) Remove the left paranthesis. [end of if structure]
7. Exit

classmate
Date
Page

A + (B * C - (D / E ↑ F) * G) * H)

	Scanned	Stack	P
	(	(	
1.	A	(	A
2.	+	(+	A
3.	(	(+(	A
4.	B	(+(	AB
5.	*	(+(*	AB
6.	C	(+(*	ABC
7.	-	(+(-	ABC*
8.	(	(+(-(	ABC*
9.	D	(+(-(	ABC*D
10.	/	(+(-(/	ABC*D
11.	E	(+(-(/	ABC*DE
12.	↑	(+(-(/↑	ABC*DE
13.	F	(+(-(/↑	ABC*DEF
14.	)	(+(-	ABC*DEF↑/
15.	*	(+(-*	ABC*DEF↑/
16.	G	(+(-*	ABC*DEF↑/G
17.	)	(+	ABC*DEF↑/G*-
18.	*	(+*	ABC*DEF↑/G*-
19.	H	(+*	ABC*DEF↑/G*-H
20.	)		ABC*DEF↑/G*-H*+

Q. A + (B * C) - D

A + BC* - D

ABC* + - D

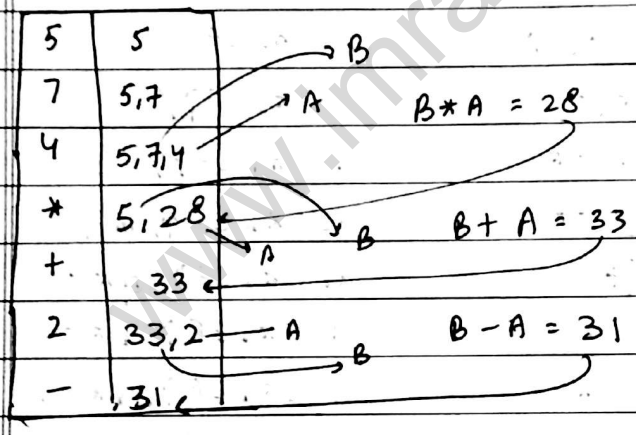
ABC* + D -

A+(B*C)-D)

	Scanned	Stack	P
1.	A	(	A
2.	+	(+	A
3.	(	(+(	A
4.	B	(+(	AB
5.	*	(+(*	AB
6.	C	(+(*	ABC
7.	)	(+	ABC*
8.	-	(-	ABC*+
9.	D	(-	ABC*+D
10.	)	∅	ABC*+D-

Postfix to infix

P: 574*+2-



P: 5,6,2,+,*,12,4,/,-

5	5
6	5,6
2	5,6,2
+	5,8
*	40
12	40,12
4	40,12,4
/	40,3
-	37

⊗ ⇒ any operator

Algorithm for
Evaluation of postfix expression to infix expression.

1. Add a ')' at the end of postfix expression (P:).
2. Scan P from left to right and repeat steps 3 and 4 for each element of P until the ')' is encountered.
3. If an operand is encountered put it on stack.
4. If an operator ⊗ is encountered then.
   (i) Remove the top two elements of stack where A is the top element and B is the next to top element.
   (ii) Evaluate B ⊗ A.
   (iii) Place the result of evaluation back on stack.
   [end of if structure] [end of step 2 loop].
5. Set value = the top element of the stack.
6. Exit.

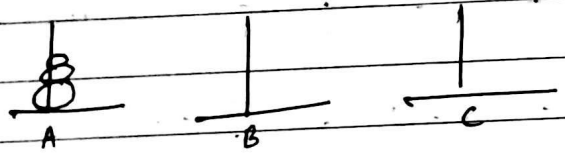
Convert postfix expression into infix

Q1. 543+2-/64+3/*

Q2. Convert postfix into prefix

Q3. Evaluate the given expression.
52C+4*6-3/+

Q4. Tower of Hanoi



2^{n-1} here n = no. of disks.

A1. P: 548+2-/64+3/*

5	5
4	5,4
8	5,4,8
+	5,7
2	5,7,2
-	5,5
/	1
6	1,6
4	1,6,4
+	1,10
3	1,10,3
/	1,3.33
*	3.33

A2. P: 526+4*6-3/+

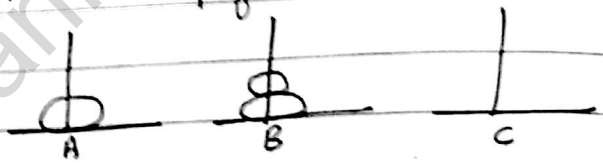
5	5
2	5,2
6	5,2,6
+	5,8
4	5,8,4
*	5,32
6	5,32,6
-	5,26
3	5,26,3
/	5,8.66
+	13.66

A3.

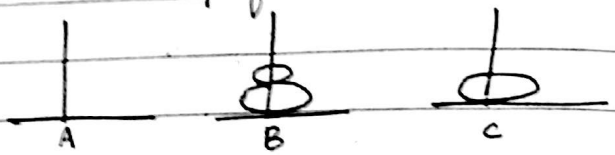
Step1. Move top of A to C.

Step2. Move top of A to B.

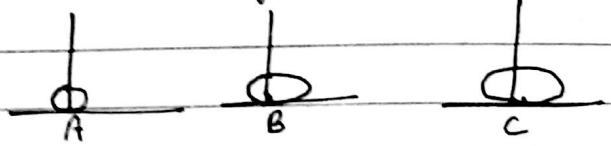
Step3. Move top of C to B.



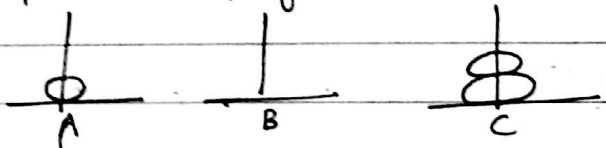
Step4. Move top of A to C.



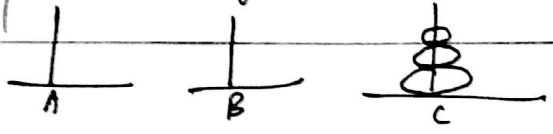
Step5. Move top of B to A.



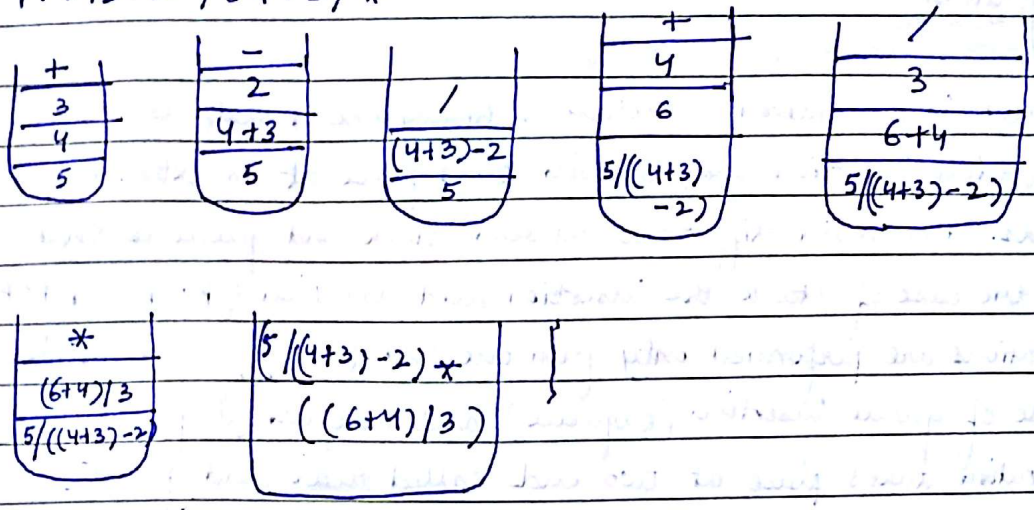
Step6. Move top of B to C.



Step7. Move top of A to C.

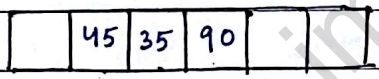


P: 543+2-/64+3/*



Queue

Q Perform the following operations on a queue of size 6 where 1st = 2 and rear = 4.



① Enqueue (65) — ① 45 35 90 65 — F=2, R=5

② Dequeue( ), Dequeue( ) — ② 35 90 65 — F=3, R=5 — ③ 90 65 — F=4, R=5

③ Enqueue (32)

④ Enqueue (95) — ④ 90 65 32 — F=4, R=6

⑤ Dequeue — ⑤ 90 65 32 — F=4, R=6. (Overflow)

⑥ 65 32 — F=5, R=6

these 4 blocks are empty but cannot be used for any enqueue operation so circular queue is used.

Algorithm

QUEUE A queue is an ordered collection of homogeneous data elements in which insertion and deletion takes place at to extreme points ends. The main difference between stack and queue is that in the case of stack the insertion and deletion operations are performed only from one end (top), whereas in case of queue insertion (enqueue) and deletion (PUSH and POP) operation takes place at two ends called rear and front.

Algorithm for insertion [enqueue (Q, N, item)] where n is length.

1. if (rear == n) then print "queue is full". exit.
2. else if (rear == 0 && front == 0)
   then set front = 1.
3. rear = rear + 1.
4. Q[rear] = item.
5. Exit.

Dequeue

1. if (front == 0) then print "Queue is empty". exit.
2. else item = Q[front].
3. if (front == rear)
   set front = 0
   rear = 0
4. else front = front + 1.
5. Exit.

Q. Write down the applications of queue in real life?

Q. Write down the algorithm for enqueue and dequeue for a circular queue?

Algorithm for insertion/enqueue into a circular queue.

① If (front = 0) then
set
(i) front = 1
(ii) rear = 1
(iii) CQ [front] = item

② else
(i) next = (Rear modulo length) + 1
(ii) if (next ≠ front)
a) Rear = next
b) CQ[rear] = item
(iii) else print "Queue is full".
(iv) end if

③ End if

④ Exit

Algorithm for Dequeue in Circular Queue

① if (front == 0) then
(i) print "Queue is empty"
(ii) Exit

② else
(i) item = CQ[front]
(ii) if (front == rear)
then set

(a) front = 0

(b) rear = 0

(iii) else

a) front = (front modulo length) + 1

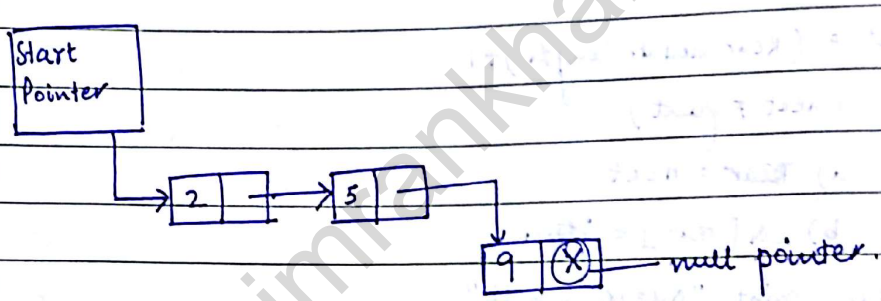
(iv) end if

3. End if

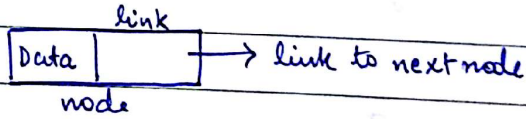
4. Exit

⇒ LINKED LIST

Singly Linked list



A linked list is called dynamic data structure where amount of memory required can be varied during its use. In linked list, adjacency between the elements are maintained by means of links or pointers. A link or pointer is the address of the ~~substitution~~ subsequent element. In linked list, data (information) and link (to point to the next data) both are required to be maintained. An element in a linked list is termed as node, which consist of two fields data and link



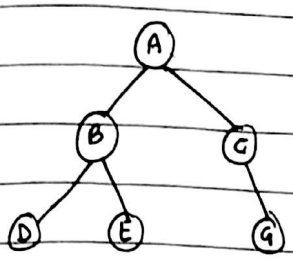
A linked list is an ordered collection of finite, homogeneous data elements called nodes where the linear order is maintained

by means of links or pointers.

Linked list can be categorised into three major groups.

1. Single linked list
2. Circular linked list
3. Double linked list

Array Representation of Tree



level 0
level 1
level 2

Block	1	2	3	4	5	6	7	8	9	10	11
	A	B	C	D	E		G				

The following rules can be used to decide the location of any node of a tree in the array starting from 1.

1. The root node is at location 1.
2. For any node with index i, $1 < i \leq n$
   (a) Parent $(i) = i/2$
   for the node $i = 1$, there is no parent.
   (b) Lchild $(i) = 2 * i$
   (c) Rchild $(i) = 2 * i + 1$

Linked Representation of Tree

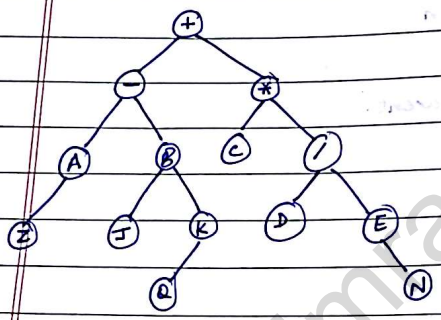
Operation on Tree

1. Insertion
2. Deletion
3. Traversing
4. searching

Traversal of Tree

There are 3 ways to traverse a tree :-

1. In order (L_T R R_T)
2. Pre order (R L_T R_T)
3. Post order (L_T R_T R)



Inorder

Z A - J B Q K + C * D / E N

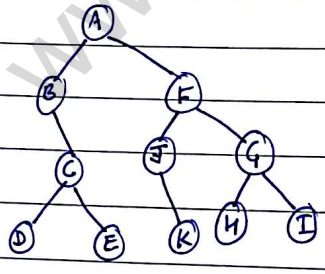
Preorder

+ - A Z B J K Q * C / D E N

Postorder

Z A J Q K B - C D N E / * +

Q.



~~D C E - B A :-~~

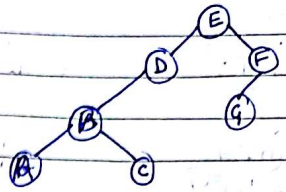
In order B D C E A J K F H G I

Preorder A B C D E F J K G H I

Postorder D E C B K J H I G F A

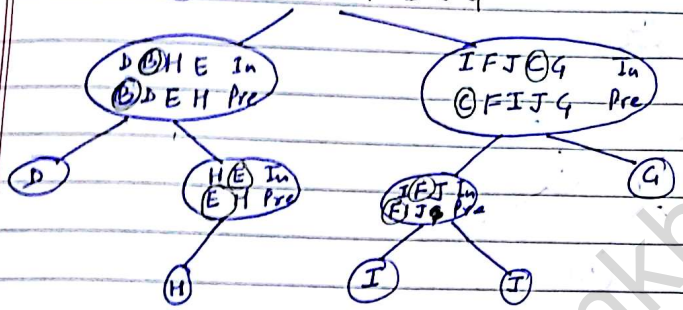
Q. A B C D E F G

Make Inorder Tree

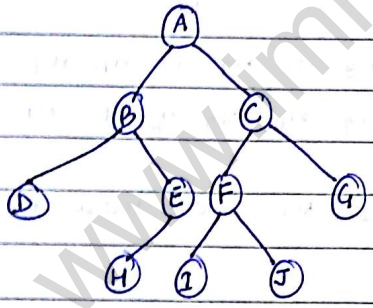


Q. Inorder D B H E A I F J C G

Preorder A B D E H C F I J G

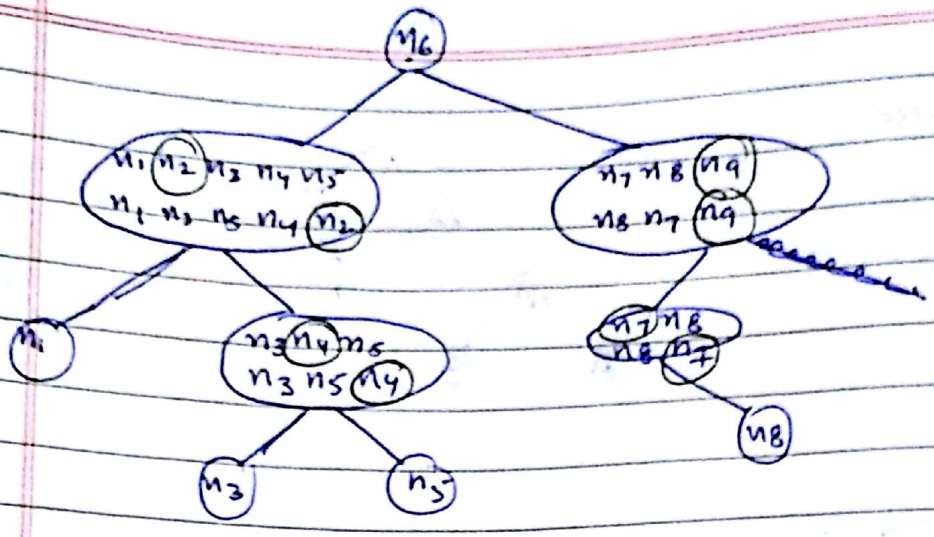


Tree

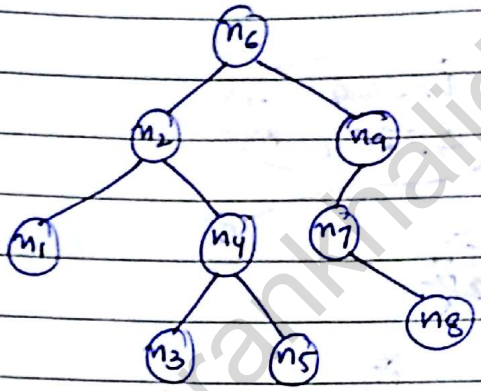


Q. Inorder n1 n2 n3 n4 n5 n6 n7 n8 n9

Postorder n1 n3 n5 n4 n2 n8 n7 n9 n6



Tree



Q. Preorder and Postorder traversal of binary tree are given as below

Preorder $n_1\ n_2\ n_3\ n_4\ n_{10}\ n_8\ n_5\ n_9\ n_6\ n_{11}\ n_7$

Postorder $n_4\ n_{10}\ n_3\ n_8\ n_2\ n_9\ n_{11}\ n_7\ n_6\ n_5\ n_1$

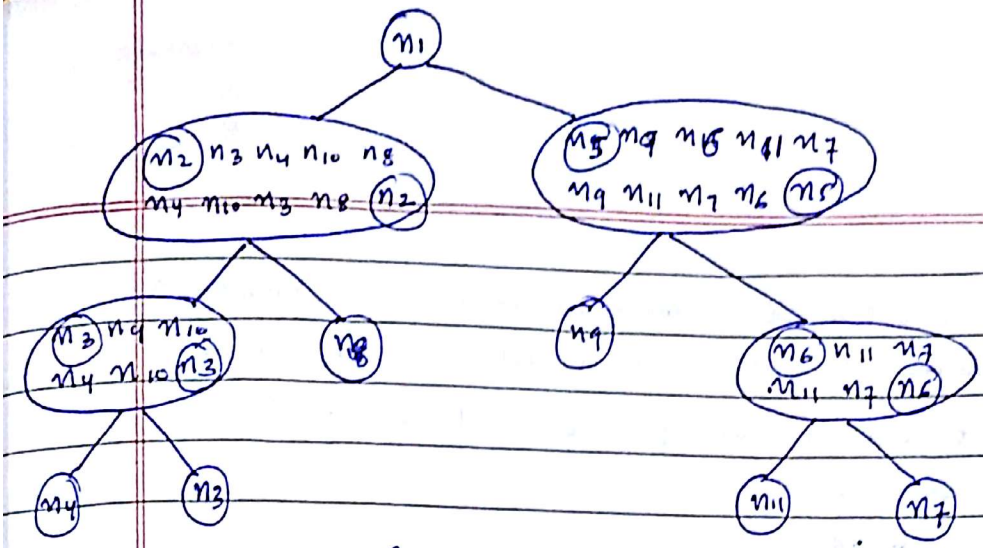
Obtain a binary tree which resembles with these traversals.

If the Inorder traversal is as follows

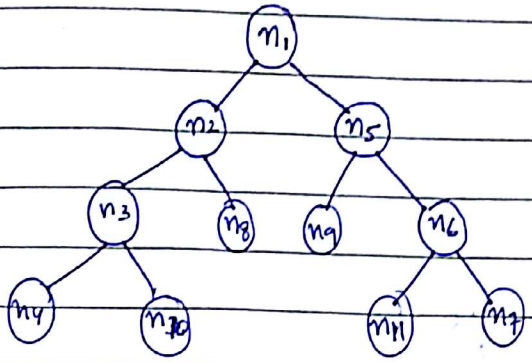
$\underbrace{n_4\ n_3\ n_{10}\ n_2\ n_8}_{L}\ \underbrace{(n_1)}_{\text{Root}}\ \overbrace{n_9\ n_5\ n_{11}\ n_6\ n_7}^{R}$

Preorder → $(n_1)\ n_2\ n_3\ n_4\ n_{10}\ n_8\ n_5\ n_9\ n_6\ n_{11}\ n_7$

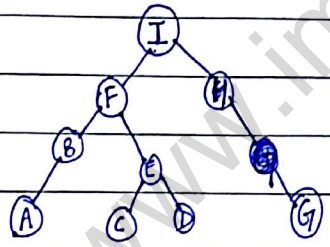
Postorder → $n_4\ n_{10}\ n_3\ n_8\ n_2\ n_9\ n_{11}\ n_7\ n_6\ n_5\ (n_1)$ — root



Tree



Q. Postorder ⇒ A B C D E F G H I



Types of Binary Tree

1. Expression Tree
2. Binary Search Tree
3. Heap Tree
4. Threaded binary Tree
5. Huffman Tree
6. Height Balanced Tree (AVL Tree)
7. Decision Tree

Expression Tree

Expression Tree is a tree which stores an arithmetic expression. The leaves are operands while the internal nodes are operators.

For example let us consider an expression and convert it into binary expression tree.

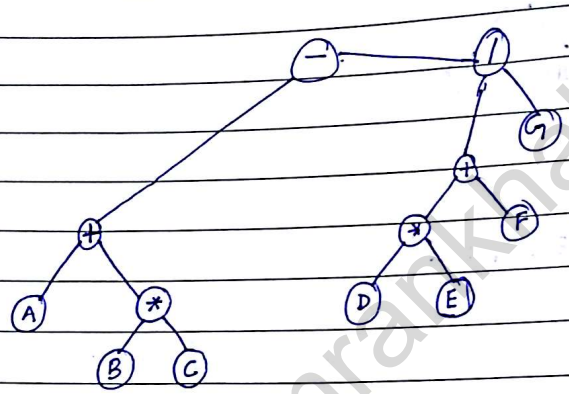
(A+B*C) - (CD*E+F)/G)

(A+BC*) - ((DE*+F)/G)

(ABC*+) - (DE*F+/G)

(ABC*+) - (DE*F+G/)

ABC*+DE*F+G/-

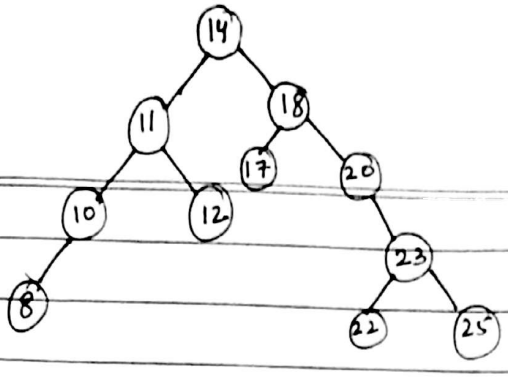


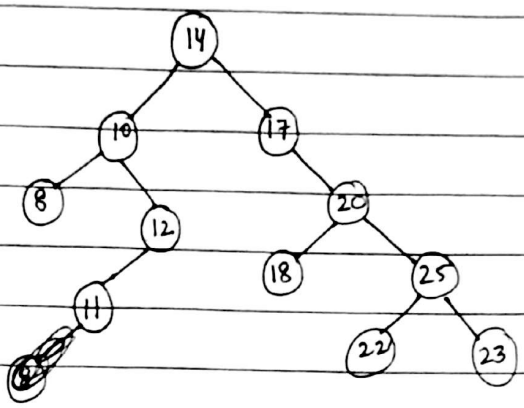
Binary Search Tree

A binary search tree T or binary sorted tree has the following properties. ~~The value of~~

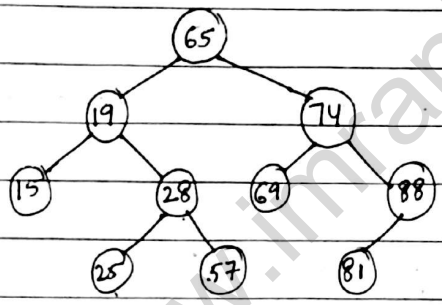
The value of ~~root~~ node ~~N~~ is greater than every value in the left sub tree and node N is less than every value in the right sub-tree of every value of node N.

14, 10, 17, 12, 10, 11, 20, 12, 18, 25, 20, 28, 22, 11, 23



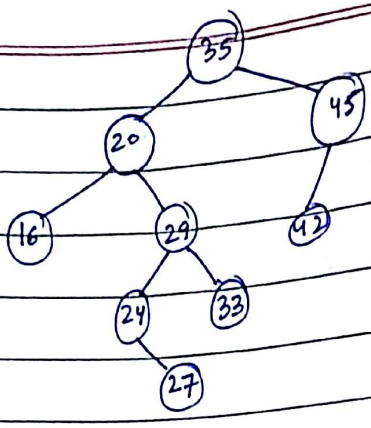


65, 19, 74, 15, 28, 25, 69, 88, 57, 81

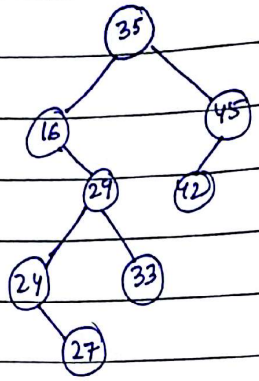


1. Searching for a data in BST.
2. Inserting any data into BST.
3. Deleting any data from BST.
4. Traversing the BST.

Q/

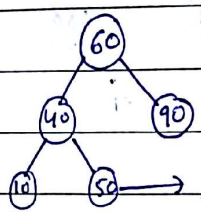


delete 20



Deleting a node from a binary search tree there can be 3 cases for deletion.

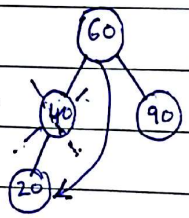
(1) n is the leaf node



50 is deleted.

(2) node has exactly one child.

40 is deleted.



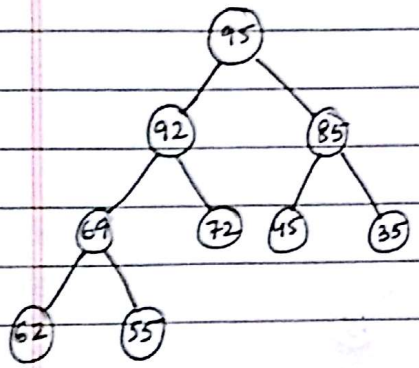
(3) node n is deleted from Tree T by first deleting successor of n from tree by using case 1 or case 2. and then replace the data content in node n by the data content in node successor n.

Heap Tree

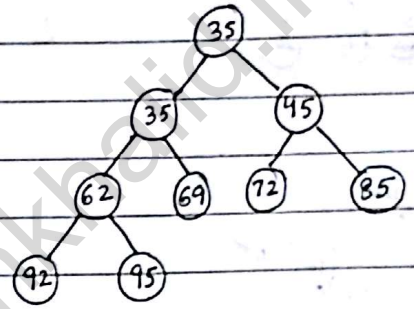
Suppose H is a complete binary tree then it will be termed as heap tree, if it satisfies the following properties

① For each node N in H the value of N is greater than or equal to the value of each of the children of N

② N has the value which is greater than or equal to the value of successor of N (max heap) such a heap tree is called max heap and min heap as exactly the opposite properties.



Max Heap Tree



Min Heap Tree

Representation of Heap Tree

35	35	45	62	69	72	85	92	95				

Operations on Heap Tree

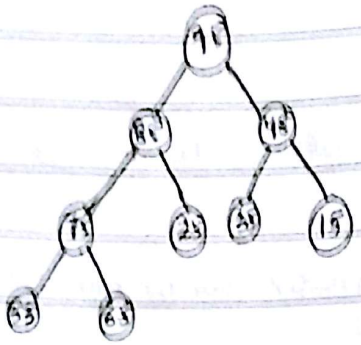
Main operations on Heap Tree are:-

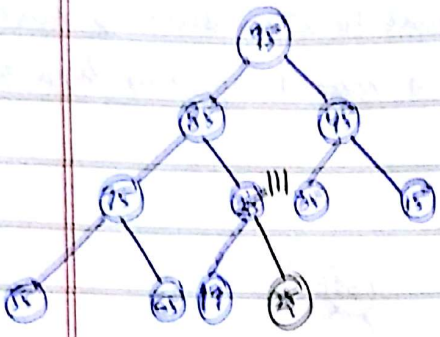
① Insertion

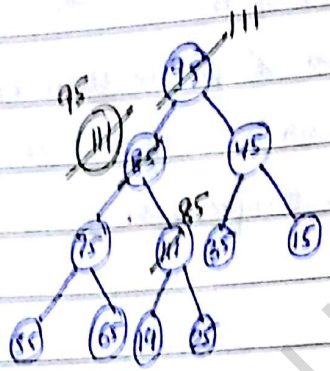
② Deletion

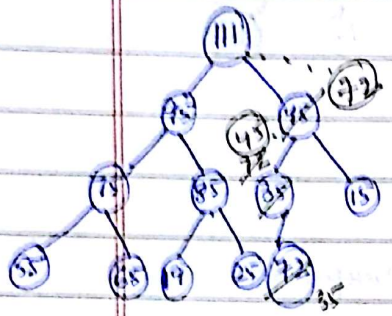
Heap Tree is mostly used in sorting

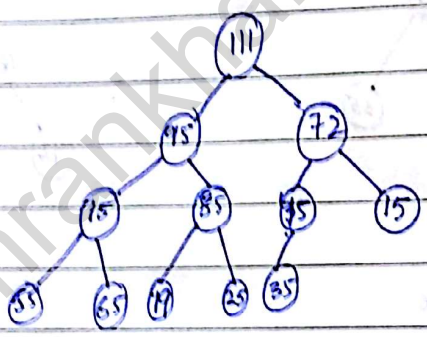
Insert 19, 111 and 72.



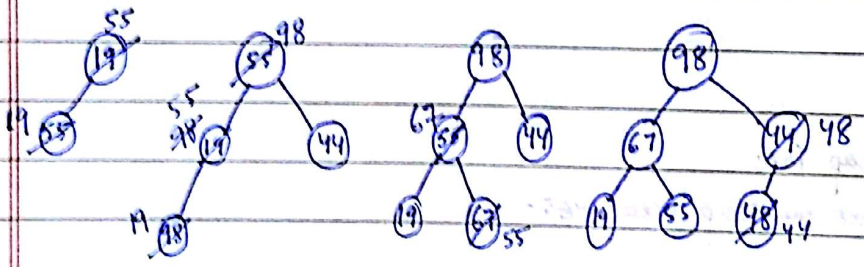


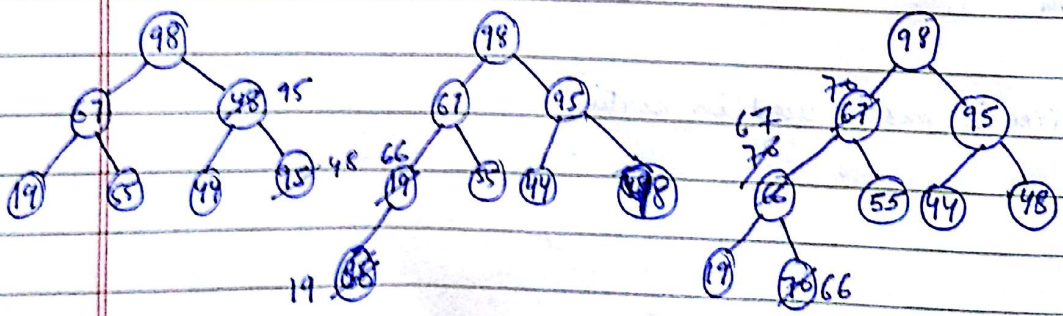


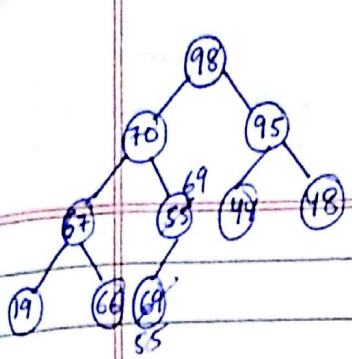


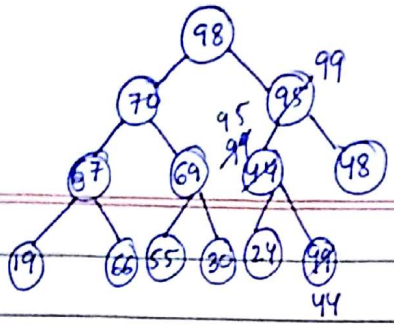


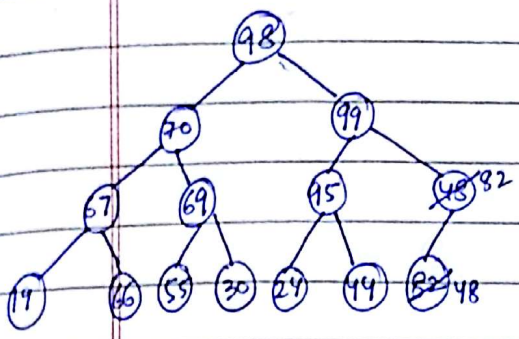
Q. Insert the following items into heap tree
19, 55, 44, 98, 67, 48, 95, 66, 70, 69, 30, 24, 99, 82.











98

Q. Build a min heap tree.

25, 65, 2, 5, 39, 36, 45, 61, 12, 10, 6, 7.

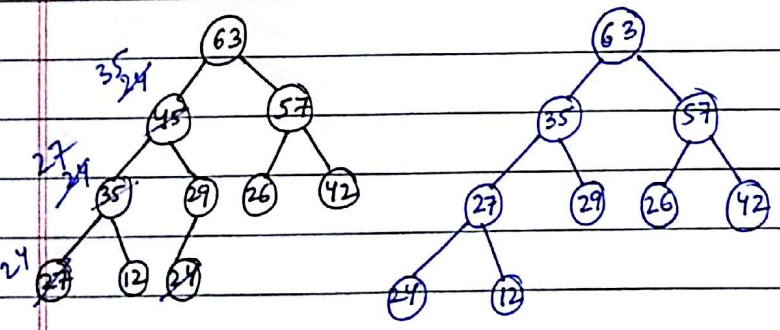
Deletion of a node from a Heap Tree.

① Delete the root node into a temporary storage say item.

② Replace the root node by the last node in the heap tree. then re-heap tree as stated below.

Let newly modified root node be the current node, compare its value with the value of its children (or 2 childs). let x be the child who's value is largest interchange the value of x with the value of current node. Make x as current node.

③ Continue re-heap if the current node is not an empty node



is a mechanism by which the processor is ...

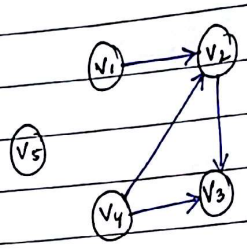
Graph

(1) Set of nodes V

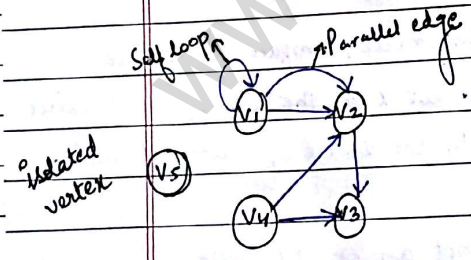
$V = \{ v_1, v_2, v_3, v_4, v_5 \}$

(2) Set of edges E

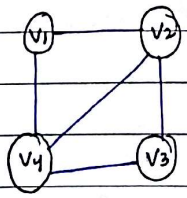
$E = \{ (v_1, v_2), (v_2, v_3), (v_4, v_3), (v_2, v_4) \}$



Nodes	v_1	v_2	v_3	v_4	v_5
Indegree	0	1	2	1	0
Outdegree	1	2	0	1	0



Cyclic and Acyclic Graph

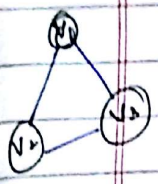


If there is a path containing 1 or more edges which start from a vertex or node V_i and terminates into the same vertex then the path is known as a cycle. If a graph or digraph does not have any cycle then it is called acyclic graph.

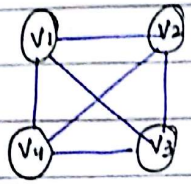
Pendent Vertex

A vertex V_i is pendent if its indegree = 1 and outdegree (V_i) = 0

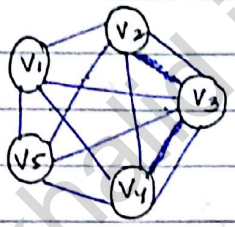
Complete Graph



3



6



10

No. of edges needed in complete graph $\frac{n(n-1)}{2}$ where n is no. of nodes

Connected Graph

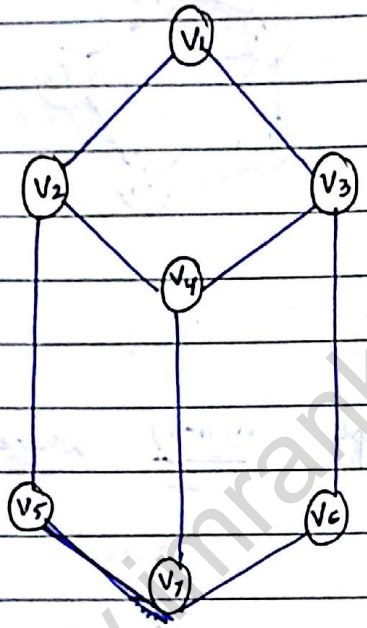
In a graph (not digraph) G, two vertices V_i and V_j are said to be connected if there is a path from V_i to V_j (or V_j to V_i). A graph is said to be connected if for every pair of distinct vertices V_i, V_j in G, there is a path

Assignment 8.1 do it in your notebook.

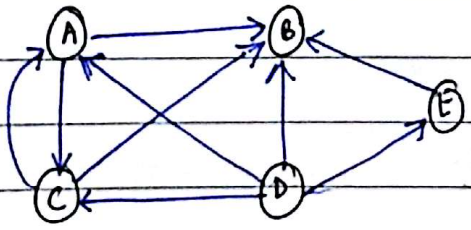
Representation of Graph

A graph can be represented in many ways. Some of the main representation

1. Set Representation
2. Link Representation
3. Sequential / Matrix Representation



Undirected graph



Directed Graph

Set Representation

G1: $V = \{ v_1, v_2, v_3, v_4, v_5, v_6, v_7 \}$

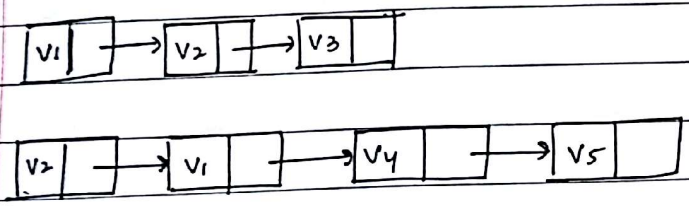
$E = \{ (v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4), (v_2, v_5), (v_3, v_6), (v_4, v_7), (v_5, v_7), (v_6, v_7) \}$

G2: $V = \{ A, B, C, D, E \}$

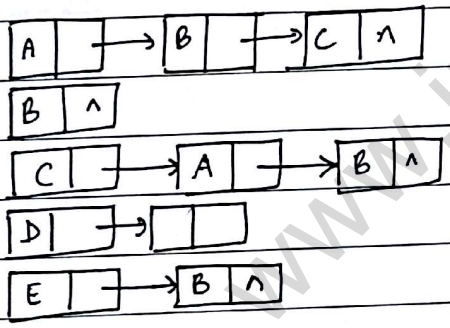
$E = \{ (A, B), (C, A), (D, C), (D, A), (C, B), (D, B), (D, E), (E, B), (A, C) \}$

Link Representation

G1:



G2:



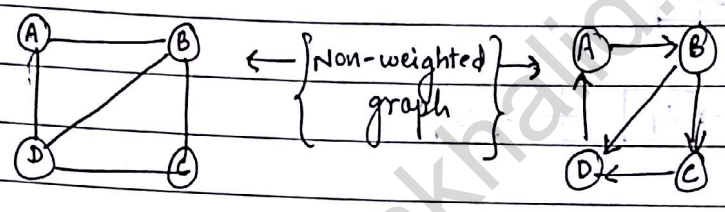
Matrix Representation

It is the most useful way of representing any graph. This representation uses a square matrix of order N x N N is the no. of vertices. Entries in the matrix is performed on the following rules.

① $a_{ij} = 1$ if there is an edge between vertex i and vertex j.
else $a_{ij} = 0$

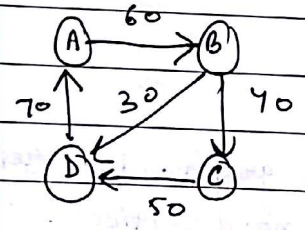
	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	1	1	0	0	0	0
V_2	1	0	0	1	1	0	0
V_3	1	0	0	1	0	1	0
V_4	0	1	1	0	0	0	1
V_5	0	1	0	0	0	0	1
V_6	0	0	1	0	0	0	1
V_7	0	0	0	1	1	1	0

8.

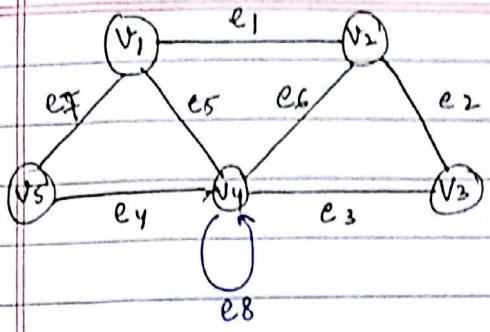


	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	1	0

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



	A	B	C	D
A	0	60	0	0
B	0	0	40	30
C	0	0	0	50
D	70	0	0	0



	v_1	v_2	v_3	v_4	v_5
v_1	0	e_1	0	e_5	e_7
v_2	e_1				
v_3					
v_4					
v_5					

Incidence Matrix

The incident matrix of graph(G) is the $m \times n$ matrix where $M=[a_{ij}]$ where $a_{ij} = 1$, if node v_i belongs to edge e_j otherwise $a_{ij} = 0$

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_1	1	0	0	0	1	0	1	0
v_2	1	1	0	0	0	1	0	0
v_3	0	1	1	0	0	0	0	0
v_4	0	0	1	1	1	1	0	2
v_5	0	0	0	1	0	0	1	0
~~v_6~~	0	0						
~~v_7~~	0	0						

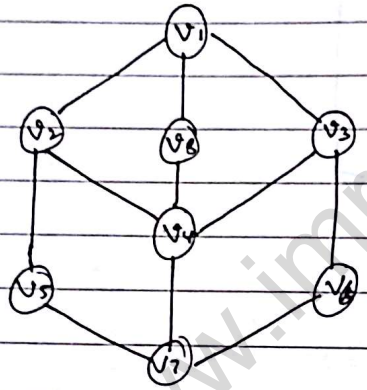
Graph Traversal

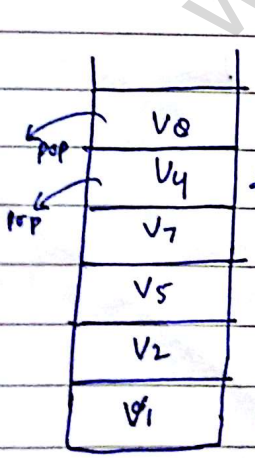
1. Breadth first search → (level by level traversal) (Queue is used)
2. Depth first search → (Inorder traversal) (stack is used)

(1) Depth First Search (DFS)

Algorithm

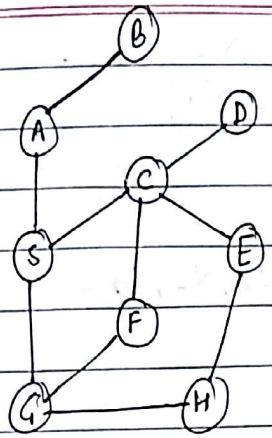
(1) PUSH the starting vertex into the stack.
(2) While (stack ! = empty)
(a) POP a vertex V
(b) if (vertex V is not in visit)
(i) visit the vertex V
(ii) store V in visit
(iii) PUSH all adjacent vertex of V on to stack.
(c) [End if]
(3) [End while]
(4) Exit

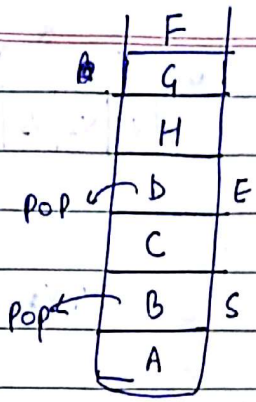




Mark V1 as visited node and put it on stack explore any other unvisited adjacent node.

Q.

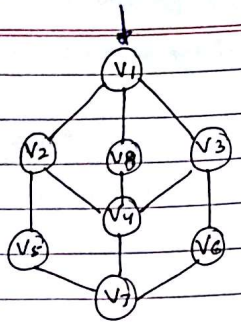


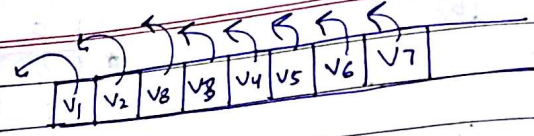


(2) Breadth first search (BFS)

Algorithm

(1) Initialise all nodes to the ready state (status = 1).
(2) Put the starting node A in Queue and change its status to the waiting state (status = 2).
(3) Repeat steps 4 and 5 until Queue is empty.
(4) Remove the front node N of Queue process N and change the status of N to the processed state (status = 3).
(5) Add to the rear of Queue all neighbours of N that are in the steady state (status = 1) and change their status to the waiting state (status = 2)
[end of step 3 loop].
(6) Exit.

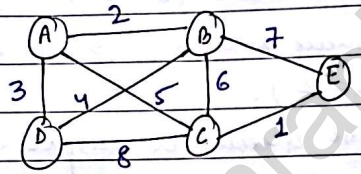


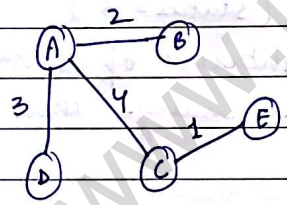


V1, V2, V8, V3, V4, V5, V6, V7

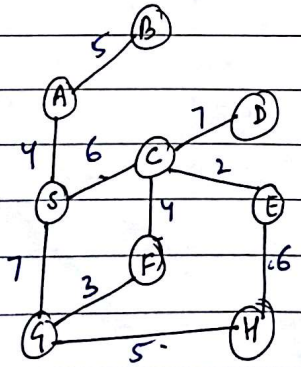
we write adjacent nodes of the vertex where front points.

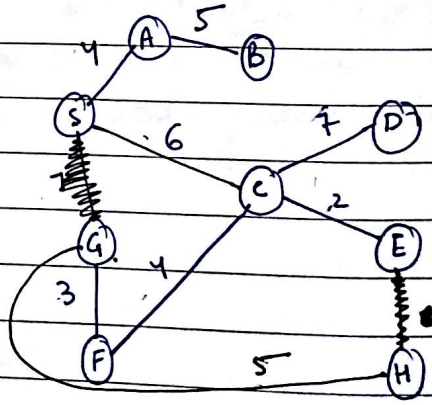
Minimum Spanning Tree from a Graph by using Kruskal's algorithm or Shortest Path Algorithm -



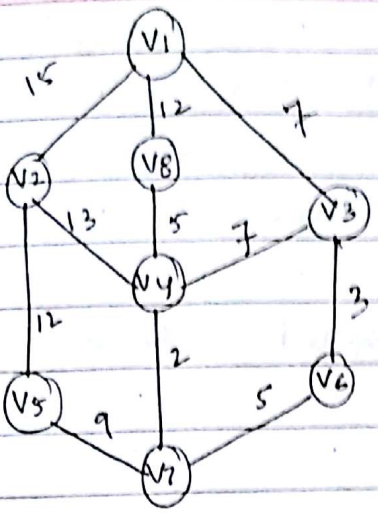


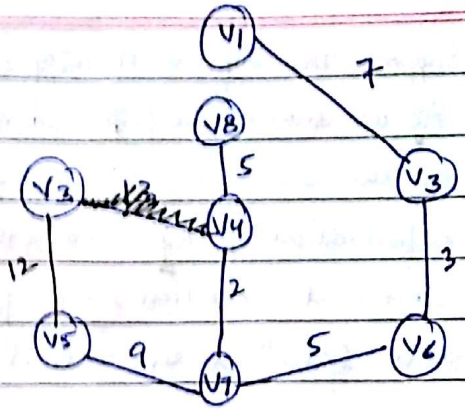
Minimum = 1+2+3+4 = 10





Minimum = ~~36~~ 34





Minimum = 43