DATA STRUCTURE AND TYPES

Collection of data in an organised manner called data Structure.

* Array
* Stacks (LIFO) Stacks of Books
* Queues (FIFO) Queue of person waiting for bus
* Linked list
* Tress
* Graphs

Array, Stacks, Queues, Linked list — Primitive

Tress, Graphs — Non-primitive

Data Structure Operation -

* Traversing (visiting the element of data structure)
* Searching
* inserting
* Deleting
* Sorting
* Merging

Algorithum - Step by Step sol^n of any operation.

Time | Space ~~trade~~ pradeoff

An algorithum is list of steps to solve a particular problem. There are two majors to find the efficiency of an algorithum.

Time and space. According to time space tradeoff, by increasing the amount of space to storing the data, one may be able to reduce the time for processing the data and vice versa.

String operation-

* Substring -

SUBSTRING (String , initial , length)

SUBSTRING('TO BE NOT TO BE', 4 , 7) = 'BE ORN'

SUBSTRING ('THE END', 4, 4) = '□END'

* Indexing

INDEX (text , pattern)

// Case Sensitive

' HIS FATHER IS THE DOCTOR'

INDEX (T, 'THE') = 7
INDEX (T, 'THEN') = 0
INDEX (T , '□THE') = 14

* Concatenation

$S_1 \,,\, || \, S_2$

$S_1 = $ 'MARK' $S_2 = $ 'TWIN'

$S_1 || S_2 = $ 'MARKTWIIN'

$S_1 || \square S_2 = $ ' MARK TWIN'

* length

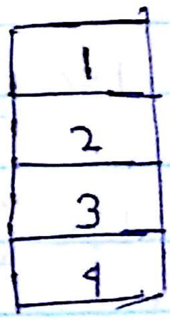
LENGTH ('COMPUTER') = 8

* Blank space included in the string.

ARRAY

A[1] A[2] . . . . A[N]

length = U·B - L·B +1



4 - 1 + 1
= 4

1D - Array -

If we want the location of element in an array

$$\text{Loc}[A[K]] = \text{Base Address} + w\,(K - L.B)$$

* Consider the linear array ~~A[5][~~ A(5:50), B(-5:10), C(18).

ⓐ Find no. of element each array (length)

Ans
$$\text{length} = U.B - L.B + 1$$
$$\text{length}(A) = 50 - 5 + 1 = 46$$
$$\text{length}(B) = 10 - (-5 + 1) = 16$$
$$\text{length}(C) = 18 - 1 + 1 = 18$$

ⓑ Base Address [A] = 300, w = 4 words per memory cell. Find the address of A[15], A[35], A[55]

$$\text{Loc}[A[15]] = 300 + 4(15 - 5) = 340$$

$$\text{Loc}[A[35]] = 300 + 4(35 - 5) = 420$$

$$\text{Loc}[A[55]] = 300 + 4(55) = \times$$

Not exist

Bubble Sort-

learn Complexity
(Searching | Sorting)

PEOPLE

No. of element = 6

Total no. of pass = 6-1 = 5

Pass-1

EPOPLE

EOPPLE

EOPPLE

~~EOPPLE~~ EOPLPE

~~EOPPEL~~ EOPLEP

Pass-2

EOLPEP

EOLEPP

Pass-3

ELOEPP

ELEOPP

Pass-4

EELOPP

Pass-5 There is no alteration

No, alteration.

Algorithum-

Lineary Search - UnSorted Array
Binary Search - Sorted Array

Linear Search

1.) Set DATA[N+1] = ITEM
2.) Set LOC = 1
3.) Repeat while DATA[LOC] ≠ ITEM
    Set LOC = LOC+1
4.) If LOC = N+1
    Search is Unsuccessful
5.) Exit

Binary Search -

(1) BEG = L.B
    END = U.B
    MID = int((BEG + END)) / 2
(2) repeat steps 3 and 4 While BEG ≤ END , DATA[MID] != ITEM

(3) If ITEM < DATA[MID]
    Set END = MID - 1
    else
    Set BEG = MID + 1

(4) Set MID = int((BEG + END)) / 2
(5) If DATA[MID] = ITEM ,
    Set LOC = MID

else

Set LOC = NULL

⑥ exit

e.g

11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99

ITEM = 40

BEG = 1, END = 13

MID = INT ((INT + END)) | 2

MID = (1+13) | 2 = 7

DATA [MID] = 55

Since 40 < 55, END = MID - 1 = 6

BEG = 1, END = 6

MID = INT (1+6) | 2 = 3

DATA [MID] = 30

Since 40 > 30, BEG = MID + 1 = 4

BEG = 4, END = 6

MID = INT((4+6)) | 2 = 5

DATA [MID] = 40

Set, LOC = 5

Q ITEM = 85,

BEG = 1, END = 13

MID = (1+13)/2

= 7

DATA[MID] = 55

85 > 55, ∴

BEG = MID+1

= 7+1 = 8

MID = (13+8)/2

= 10

DATA[MID] = 77

85 > 77

BEG = MID+1

= 11

MID = (11+13)/2

= 12

DATA[MID] = 88

~~88 ≥~~ 85 < 88

∴

LOC = NULL

limitation - * the list must be sorted

* One must have a direct acess to middle element in any sublist.

V.Imp

Sparse MATRICES-

Matrices with a relatively high proportion of 0 entry are called Sparse matrices.

It is usually nxn matrices, e.g

$$\begin{bmatrix} 4 & & & & \\ 3 & -5 & & & \\ 1 & 0 & 6 & & \\ -7 & 8 & -1 & 3 & \\ 5 & -2 & 0 & 2 & -8 \end{bmatrix}$$

Triangular Sparse Matrix

5X5

$$\begin{bmatrix} 5 & -3 & & & & & \\ 1 & 4 & 3 & & & & \\ & 9 & -3 & 6 & & & \\ & & 2 & 4 & -7 & & \\ & & & 3 & 1 & 0 & \\ & & & & 6 & -5 & 8 \\ & & & & & 3 & 1 \end{bmatrix}$$

7X7

Tridiagonal Matrix

aminotes

5 2-D Array - (Array of Array)

M - no. of rows
N - " " column

J - lower bound
K - upper bound
unknown matrices

Colomn major order

$$Loc(A[J,K]) = Base(A) + W(M(K-1) + (J-1))$$

M → no. of rows

Row major order

$$Loc(A[J,K]) = Same(A) + W[(N(J-1) + (K-1))]$$

N → no. of coloums

Q Consider 25X4, Suppose Base Addres = 200, W = 4 words per memory cell. Let this 2D uses row major order, Find the adress of A[2,3]

$$= 200 + 4(4[2-1) + (3-1])$$
$$= 224$$

Multidimension array -

$$Loc(C[K_1, K_2 \cdots K_N])$$

$$Base(C) + W[(((\cdots (E_N L_{N-1} + E_{N-1}) L_{N-2})$$
$$+ \cdots + E_3) L_2 + E_2) L_1 + E_1)$$

Effective indices $= E_i = K_i - LB$

Row major

$$\text{Base}(C) + W\left[\left(\cdots\cdots\left(\left(E_1L_2+E_2\right)L_3+E_3\right)L_4+\cdots\cdots E_{N-1}\right)L_N+E_N\right]$$

Imp

Suppose a Multidimensional array is declared

$A(-2:2,\ 2:22)$

$B(1:8,\ -5:5,\ -10:5)$

(a) Find the length of each dimension and find no. of each element.

A	B
$L_1 = 2-(-2)+1 = 5$	$L_1 = 8$
$L_2 = 22-(2)+1 = 21$	$L_2 = 11$
	$L_3 = 16$

no. of element $A = 5 \times 21 = 105$

$B = 8 \times 11 \times 16 = 1408$

(b) Consider the element $B[\overset{K_1}{3},\overset{K_2}{3},\overset{K_3}{3}]$. Find the effective indices E_1, E_2, E_3 $\quad K_i - LB$

$E_1 = 3-1 = 2$

$E_2 = 3-(-5) = 8$

$E_3 = 3-(-10) = 13$

(c) Find the add of element B[3,3,3] assuming the base adres = 400, $\omega = 4$.

Ans let us assume B is stored Colomn major order.

$$E_3 L_2 = 13 \times 11 = 143$$

$$E_3 L_2 + E_2 = 143 + 8 = 151$$

$$(E_3 L_2 + E_2) L_1 = 151 \times 8 = 1208$$

$$(E_3 L_2 + E_2) L_1 + E_1 = 1208 + 2 = 1210$$

$$\text{loc } B[3,3,3] = 400 + 4(1210)$$

$$= 400 + 4840$$

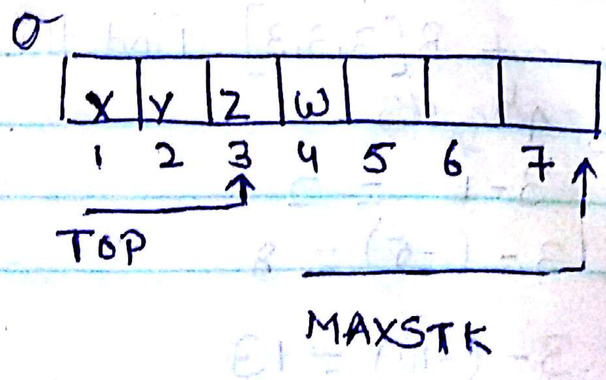
$$= 5240$$

Module-2

Stacks (LIFO)

* Stacks → Push | Pop

Array Representation of Stacks →



Push ω

top++

Algorithum -

Push (insert)

1.) If TOP= MAXSTK, then print OVERFLOW and return.

2.) Set TOP= TOP+1

3.) Set STACK [TOP] = ITEM

4.) Exit.

Pop (Delete)

1.) If TOP= 0, then print UNDERFLOW and return

2.) Set ITEM = STACK [TOP]

3.) Set TOP= TOP-1

4.) exit.

Q Consider the Stack of characters, where Stack is allocated N=8 memory cells. Describe the Stack as the following expression take place

STACK: A C D F K _ _ _

(a) POP (STACK, ITEM)

A C D F K _ _ _ _

(b) POP (STACK, ITEM)

A C D _ _ _ _ _

aminotes

(c) PUSH (STACK , P)
A C D L P _ _ _

(d) POP (STACK , ITEM)
A C D L _ _ _ _

When will overflow occur
After 8^{th} element

(b) When will C be deleted before D.
Not Possible. C

2.) N = 6 , Stack : AAA , BBB , EEE , GGG , _ ,
(a) Push Stack , KKK
AAA , BBB, EEE , GGG , KKK ___

(b) Pop
AAA , BBB , EEE , GGG ___ , ___

V.V. Imp

Arthimetic Operation | Polish Notation.

Prefix Notation , Infix Notation , Postfix Notation.
+ AB A + B AB +

Precedence

$\uparrow$ (power), /, $*$, +, -

Convert the following infix expression into prefix

(i)

$(A+B)*C$

$[+AB]*C$

$*+ABC$

(ii) $A+(B*C) = A+[*BC] = +A*BC$

(iii) $(A+B)/(C-D) = [+AB]/[-CD] = /+AB-CD$

(iv) transite, by inspection and hand, each infix expression into its equivalent postfix expression.

(a) $(A-B)*(D/E)$

$(AB-)*(DE/)$

$AB-DE/*$

(b) ~~(A+B)~~ $(A+B\uparrow D)/(E-F)+G$

$(A+BD\uparrow)/(EF-)+G$

$(ABD\uparrow)+/(EF-)+G$

$[ABD\uparrow+EF-/]+G$

$ABD\uparrow+EF-/G+$

(c) A * (B+D) / E-F * (G+H/K)

~~A * (BD+) / (EF-) * (G+HK/)~~

~~(ABD+*) / (EF-) * (G+HK/)~~

~~(ABD+*EF-/) * (G+HK/)~~

~~(ABD+*EF-/) * (GHK/+)~~

~~ABD+*EF-/ GHK/+*~~

G+H/K
G+HK/
GHK/+

A * [BD+] / E - F * [GHK/+]

A * [BD+E/] - F * [GHK/+]

[ABD+E/*] - [FGHK/+*]

ABD+E/*FGHK/+*-

Evolaution of a postfix expression-

Step-1 Add the ")" at the end of the expression

Step-2 Scan the expression from left to Right and repeat steps 3 and 4 for each element Until right porentis ")" is encounter.

Step-3 If an oppond is ecounter, put it onto Stack.

Step-4 If an operator is encounter, then

(a) Remove the top two elements of stack where 'a' is top element 'b' is the next to top element.

aminotes

(b) Evaloule 'B ⊗ A

(c) place the result of part b back to stack.

(d) Set Volue = top element on stack.

(e) exit

(1) Consider the following arthimetic expression into postfix expression

(a) P: 5, 6, 2, +, *, 12, 4, /, —

Symbol Scanned	STACK
5	5
6	5, 6 (b)
2	5, 6, 2 (c)
+	5, 8
*	40
12	40, 12
4	40, 12, 4
/	40, 3
—	37
)	

(b) P: 12, 7, 3, -, /, 2, 1, 5, +, *, +

(i) Translate P by inspection and hand into its equivalent infix expression

Symbol Scanned	STACK
12	12
7	12, 7
3	12, 7, 3
-	12, 4
/	3
2	3, 2
1	3, 2, 1
5	3, 2, 1, 5
+	3, 2, 6
*	3, 12
+,)	15

Using algo

(ii) Evaluate the infix expression

(iii) evaluate P. using algorithum

(i) Inspection and hand

P: 12, 7, 3, -, /, 2, 1, 5, +, *, +

12, (7-3), /, 2, 1, 5, +, *, +

12 / (7-3), 2, 1, 5, +, *, +

12 / (7-3), 2, (1+5), *, +

12 / (7-3), 2 * (1+5), +

[12 / (7-3)] + [2 * (1+5)]

On evaluation 3 + 12 = 15

aminotes

1

Transforming infix to postfix Expression.

Algorithum →

1.) Push a left parenthis "(" on to stack and add right parenthesis to the end of expression.

2.) Scan the expression from left to right and repeat Steps 3 to 6 for each element until the stack is empty.

3.) If an operand is ecounter, added to P

4.) If a left is parenthis "(" ecounter, push it onto stack.

5.) If an operator is ecounter, then

ⓐ Repeadely Pop from Stack and add to P each operator which has the same precedence or higher precedence than operator.

ⓑ Add operator to stack.

⑥) If a right ")" ecounter

ⓐ Pop from stack and add to P each operator until the left parenthesis ecounter.

ⓑ Remove the left parenthesis "("

⑦ Exit

(+ (* -) ×
II I
(+ (- * ✓

① A + (B * C - (D / E ↑ F) * G) * H

Symbol Scanned	STACK	Expression P
A	(	A
+	(+	A
(	(+ (	A
B	(+ (	A B
*	(+ (*	A B
C	(+ (*	A B C
-	(+ (-	A B C *
(	(+ (- (	A B C *
D	(+ (- (	A B C * D
/	(+ (- (/	A B C * D
E	(+ (- (/	A B C * D E
↑	(+ (- (/ ↑	A B C * D E
F	(+ (- (/ ↑	A B C * D E F
)	(+ (-	A B C * D E F ↑ /
*	(+ (- *	A B C * D E F ↑ / G
G	(+	A B C * D E F ↑ / G * -
)		A B C * D E F ↑ / G * - H * +
*		
H		
)		

Application of Stacks

1st element - pivot element

(1) Quick sort

44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66

← R to L

22, 33, 11, 55, 77, 90, 40, 60, 99, 44, 88, 66 (Smaller than 44)

→ Left to right (greater than 44)

22, 33, 11, 44, 77, 90, 40, 60, 99, 55, 88, 66

← R to L

22, 33, 11, 40, 77, 90, 44, 60, 99, 55, 88, 66

→ L to R

$\underbrace{22, 33, 11, 40}_{\text{I sublist}}$, 44, $\underbrace{90, 77, 60, 99, 55, 88, 66}_{\text{II sublist}}$

11, 33, 22, 40

11, 22, 33, 40

90, 77, 60, 99, 55, 88, 66 ← R to L

66, 77, 60, 99, 55, 88, 90

→ L to R

66, 77, 60, 90, 55, 88, 99 ← R to L

66, 77, 60, 88, 55, 90, 99

55, 77, 60, 88, 66

→ L to R

55, 66, 60, 88, 77

55, 60, 66, 88, 77

77, 88

1. Set Left = BEG, RIGHT = END, and LOC = BEG
2. Scan from right to left. ~~repeat~~
   a. Repeat while a[loc] <= a[RIGHT]
      right = right-1
   b. If loc = right, then return
   c. If a[loc] > a[right]
      temp = a[loc]
      a[loc] = a[right]
      a[right] = temp
   d. Set loc = right
   e. Go to Step 3
3. Scan from left to right
   a. Repeat while a[left] <= a[loc]
      left = left +1
   b. If loc = left, return
   c. If a[left] > a[loc]
      temp = a[loc]
      a[loc] = a[left]
      a[left] = temp
   d. Set loc = left
   e. Go to step 2

V.V.V. Imp

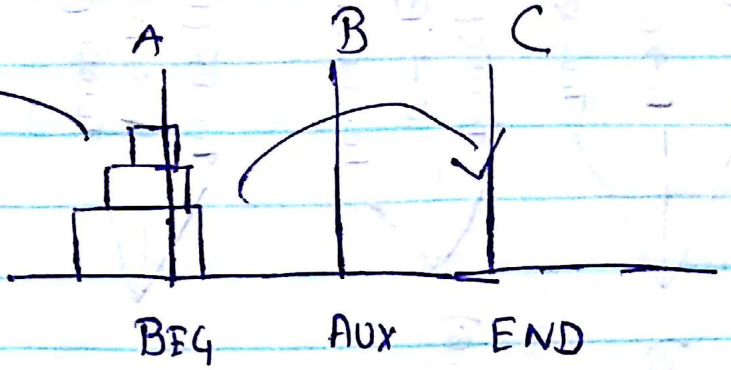
Tower of Hanoi -

N = No. of discs

Moves = $2^N - 1$

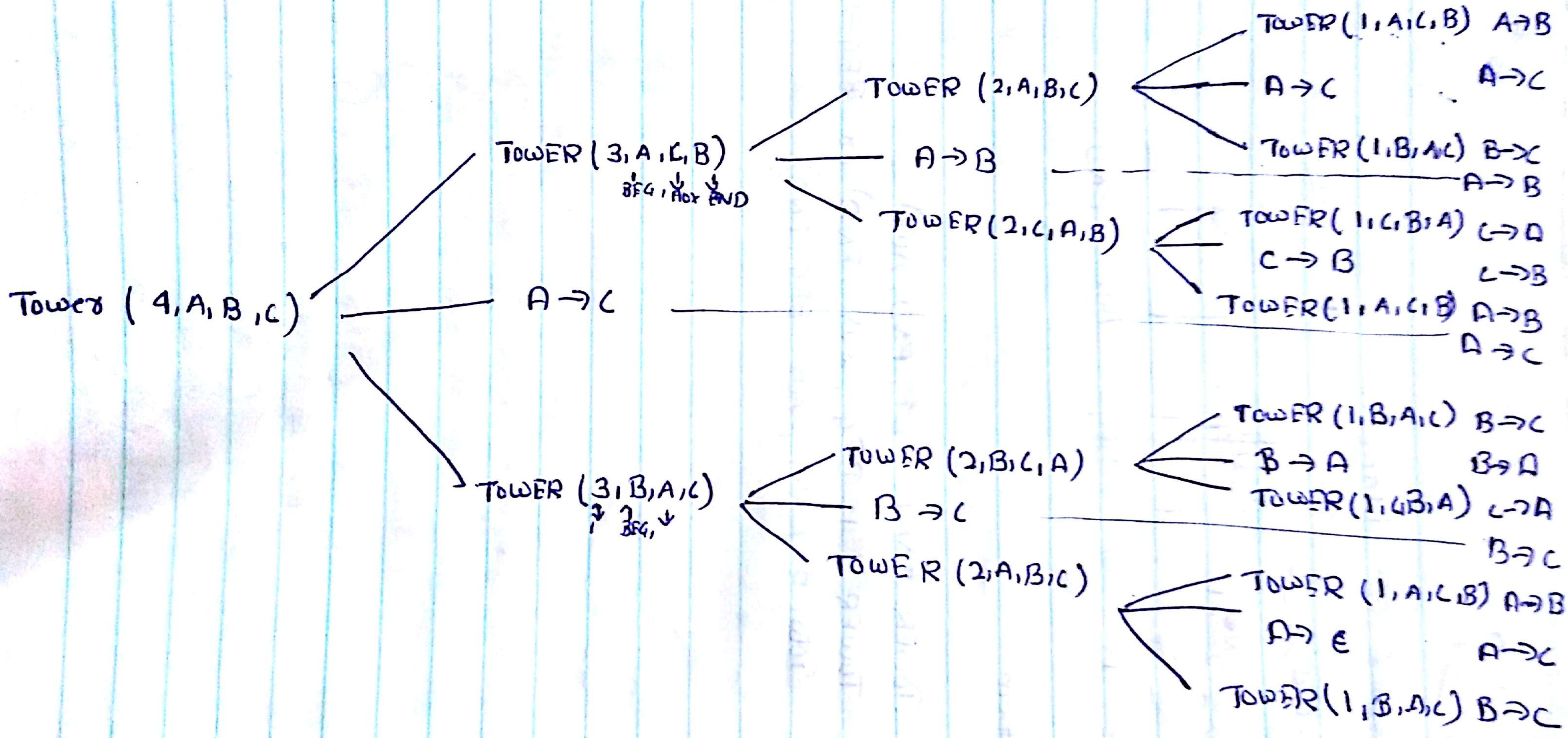
for 3 = 7

for 4 = 15



Algorithum

1. TOWER (N-1, BEG, END, AUX)
2. TOWER (1, BEG, AUX, END) or BEG → END
3. TOWER (N-1, AUX, BEG, END)


Tower (4, A, B, C)
TOWER (3, A, C, B)
BEG, AUX, END
TOWER (2, A, B, C)
TOWER (1, A, C, B) A→B
A→C
A→C
TOWER (1, B, A, C) B→C
A→B
A→B
TOWER (2, C, A, B)
TOWER (1, C, B, A) C→A
C→B
C→B
TOWER (1, A, C, B) A→B
A→C
A→C
TOWER (3, B, A, C)
BEG,
TOWER (2, B, C, A)
TOWER (1, B, A, C) B→C
B→A
B→A
TOWER (1, C, B, A) C→A
B→C
B→C
TOWER (2, A, B, C)
TOWER (1, A, C, B) A→B
A→C
A→C
TOWER (1, B, A, C) B→C

Queues - (FIFO)

It is first in first out, deletion can take place from one end called front and insertion at rear.

ⓐ Initially empty
front = rear = 0

N=5

1	2	3	4	5

Queue

ⓑ Insert A, B, and C
front = 1
rear = 3

A	B	C		
1	2	3	4	5

ⓒ A deleted
F=2
R=3

	B	C		
1	2	3	4	5

ⓓ Insert D and E
F=2
R=5

	B	C	D	E

ⓔ B and C deleted
F= 4
R= 5

			D	F
1	2	3	4	5

(f) Insert F

F = 4
R = 1

1	2	3	4	5
F			D	E

(g) Delete D

F = 5
R = 1

F				E

(h) Insert G and H

F = 5, R = 3

F	G	H		E

(i) Delete E

F = 1, R = 3

F	G	H		

Algorithum - (same for circular Queue)

(1) Queue insertion

(a) If front = 1 and rear = N or front = rear + 1 (C·Queue)
write "overflow" and exit

(b) If front = NULL
~~write unde~~ front = 1
rear = 1

else if
rear = N, then (circular queue)
set rear = 1

else

set rear = rear + 1

(c) set queue[rear] = item.

(d) exit.

(ii) Queue deletion

(a) If front = rear = 0

write 'underflow' and exit.

(b) set item = Queue [front]

(c) If front = rear

then set front = rear = 0

else if

front = N (C.Queue)

set front = 1

else

set front = front+1

(d) exit.

Insert	Deletion
* Full element	* No element
* No element	* 1 element.

condition to check.

Dequeues (double ended ~~nod~~ queue)

* Input restrict deque allow insertion at only one end of the list but allows deletion at both end of the list.

* Out restricted deque alow deletion at only one end of the list but allows insertion at both end of the list.

front → left
rear → Right

Priority Queues -

Rule-1 An element of higher priority is processed before any element of lower priority.

Rule-2 Two element with the same priority are processed according to the order in which they were added in queue.

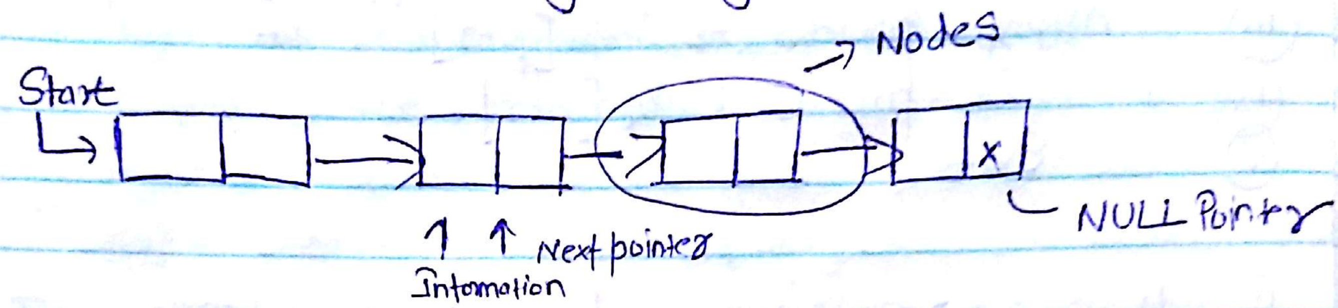
⇒ Array Representation

		1	2	3	4	5	6
F=1, R=2	1		A				
F=1, R=3	2	B	C	X			
F=0, R=0	3						
F=5, R=1	4	F				D	E
F=R=4	3				G		

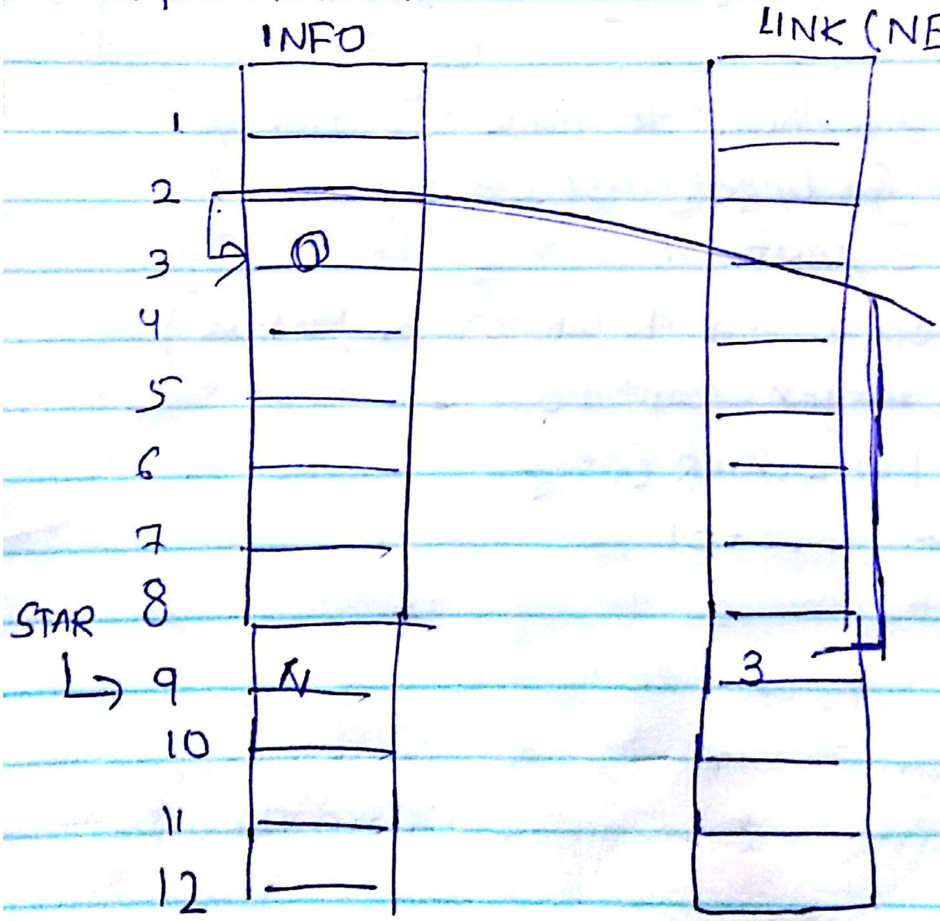
Module-3

Linked list

It is a linear collection of data called nodes where the linear order is given by means of POINTERS.



Representation →



START = 9

INFO [9] = N

LINK [9] = 3

INFO [3] = O

Traversing a linkedlist

(I) Set ptr = START
(II) Repeat step 3 and 4 while (ptr != NULL)
(III) Apply process to info [ptr]
(IV) Set ptr = link [ptr]
(V) Exit

Q Write an algorithum to print the information at each of linked-list
write INFO [ptr]

Q Write an algorithum to find the no. of element in linked-list.
(I) Set ptr = START
(II) Repeat step 3 and 4 while (ptr != NULL)
(III) Take integer count = 0
(IV) Set ptr = link [ptr]
(V) Set count += 1
(VI) exit

aminotes

⇒ Searching

ⓐ When list is unsorted

(i) Set ptr = START

(ii) Repeat Steps while (ptr != NULL)

(iii) if item == info[ptr]

Set loc = ptr

exit

(iv) else ptr = link[ptr]

(v) Set loc = NULL (Search unsuccessful)

(vi) exit.

ⓑ Sorted list

(i) Set ptr = START

(ii) Repeat Steps while (ptr != NULL)

(iii) if item < info[ptr]

exit

else if item == info[ptr]

loc = ptr

exit

else if item > info[ptr]

ptr = link[ptr]

else loc = NULL

(iv) Exit.

Memory Allocation / Garbage Collection

A Special linked list is maintained which consist of unused memory cell. If I want to insert few nodes, i fetched these new nodes from unused memory. Similary when i delete these nodes, deleted nodes goes to unused memory and available for future used. This linked list has its own pointer called freeStorage list or free pool.

Link [info, LINK, START, AVAIL]

Garbage Collection - OS of a computer periodically collects all the deleted space onto the free storage list. This technique is called G. collection.

(I) Computers runs through all the list, tagging those cell which are in used.
(II) Computer runs through the memory collecting all untag space onto the free storage list.

Inserting at a begining of list.

1.) If avail = NULL , overflow exit.

2.) Set new = Avail and avail = link [avail]

3.) Set info [new] = item

4.) Set link [new] = Start

5.) Set Start = new

6) exit.

⇒ inserting after a given node.

1.) If avail = NULL
write overflow and exit.

2.) Set new = avail
link = link [avail]

3) Set info [new] = item

4) if loc = null , then
Set link [new] = Start
Start = new
else
set link [new] = link [loc]
link [loc] = new

5) exit.

aminotes

⇒ inserted into sorted linked list.

1) If avail = NULL
overflow and exit.
2.) Set new = avail.
3.) Set info [new] = item

node following given nod
⇒ Deletion of linked list

1. If locp = null
then Set start = link [start]
else
Set link [locp] = link [loc]

2. Set link [loc] = avail
avail = loc

3. exit

Deleting the node with a given item of info.

(1) When Start = NULL
Underflow and exit.

(3) if locp = null
set Start = link[start]
else
set link [locp] = link [loc]

(2) While (ptr = Start)
if info [loc] = item

if loc = NULL
(4) link[locp] = null

(5) link [loc] = avail
avail =

6) exit.

Header linklist ->
It's a link list which contain a special node called header node at the begining of the list. It is two types -

* A grounded header list where the last ~~zero~~ node contain the null pointer.

* A circular header list where the last node point backs to the header node.

Triversing circular link list.

1. let ptr = link of start
2. Repeat steps 3 and 4 while ptr ≠ start.
3. Apply process to info of ptr.
4. Set ptr = link of ptr.
5. exit.

* Every node has three parts -
  i. Co-efficient
  ii. Exponent
  iii. The link pointer.

link representation of Stacks -

Start = top

e.g. x, y, z, a, b

Push ω

ω, x, y, z, a, b

Pop x

ω, y, z, a, b

Algo-

Push in a link stack

1. If avail = NULL
   write overflow and exit.
2) Set new = Avail and Avail = link [Avail]
3) Set info[new] = item
4) Set link[new] = top
5) Set top = new
6.) exit.

pop ~~of~~ in a link Stack

1. If top = NULL
   write underflow and exit.
2. Set item = info[top]
3) Set temp = top and top = link [top]
4) Set link [temp] = avail and avail = temp
5) exit.

Link Representation of Queues

a, b, c, d

Insert

a, b, c, d, e

Delete a from queue

b, c, d, e

(1) Insertion

(i) If avail = NULL
write overflow and exit.

(ii) Set new = Avail and Avail = link [Avail]

(iii) Set info [new] = item and link [new] = NULL

(iv) If front = NULL then front = rear = new
else set link [rear] = new and rear = new

(v) Delete

(II) Deletion

(i) If front = NULL
write underflow and exit.

(ii) Set temp = front

(iii) item = info [temp]

(iv) front = link [temp]

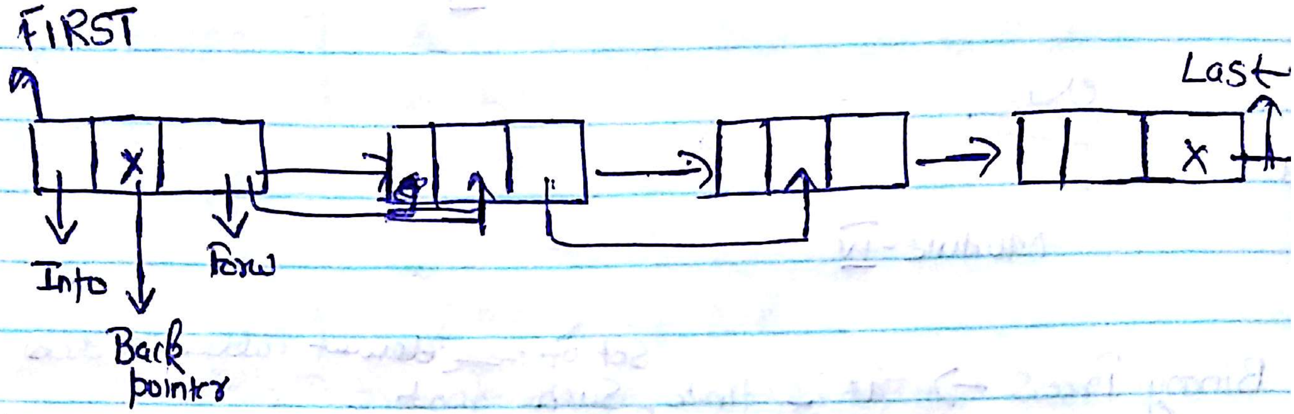
(v) link [temp] = Avail, Avail = temp

(vi) exit.

Press - linked list (V. Imp)

Algorithm = procedure | Pseudo code

Two-way linked list | Double linked list -



(i) Transcerving - | Searching | Deletion

(i) Set forw[Backward[loc]] = forward[loc];
and backward [forward [loc] = backward [loc];

(ii) Set forw[loc] = avail ;
avail = loc ;

exit.

(ii) Insertion -

Set new = avail
avail = forwar[avai]
info (new) = item

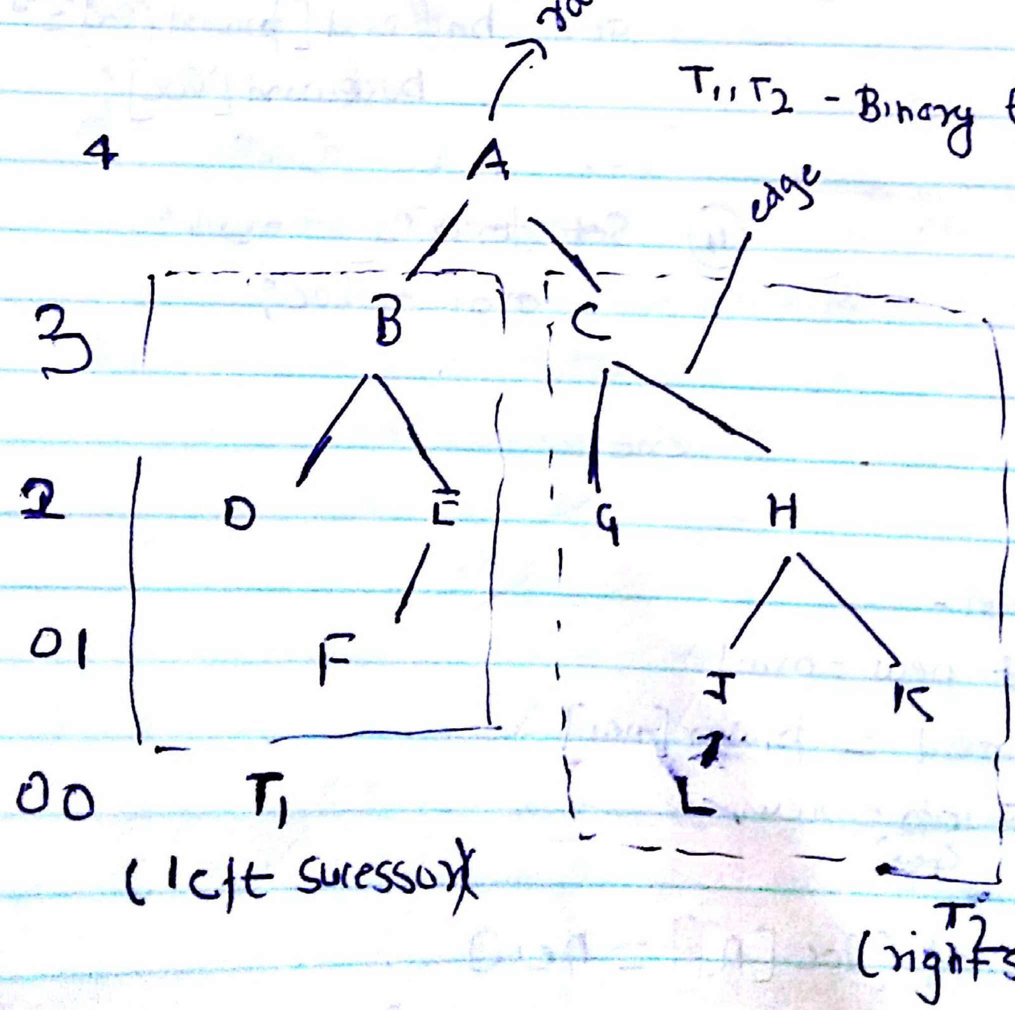
forward (loc [A]] = new
forward [new] = loc [B]

backward [loc] = B[new]
back [new] = loc [A]

exit.

Module-IV

Binary Trees → It is finite Set of ~~node~~ element called nodes Such that
(a) T is empty
(b) Tress contain a distingude node R, called the root of T and remaing nodes of T form order pair of disjoint tress T_1 and T_2.



T_1, T_2 - Binary tress
└→ Nodes (0, 1, 2)

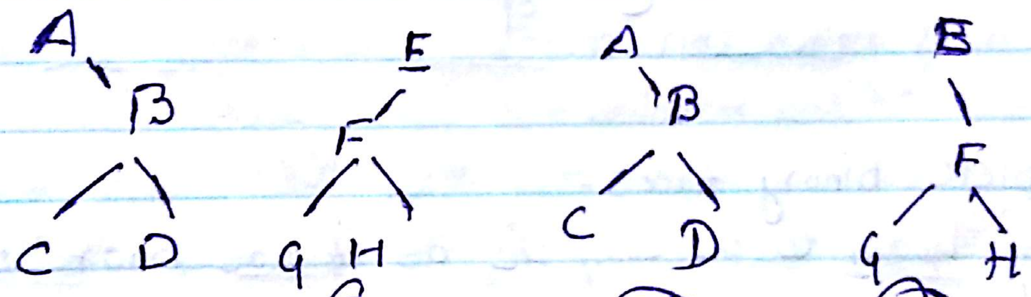
path- Collection of edge to transverse one node to another node

Termina nodes
| leaf nodes - has zero parent (D, F, G, L, K)

edge] A-C
C-H
H-K

| path - A-C-H-K

Height of a tree - | Depth of tree
= max^m no. of node in a branch
A-C-H-J-L
height = 5



a b c d

Similar a c d

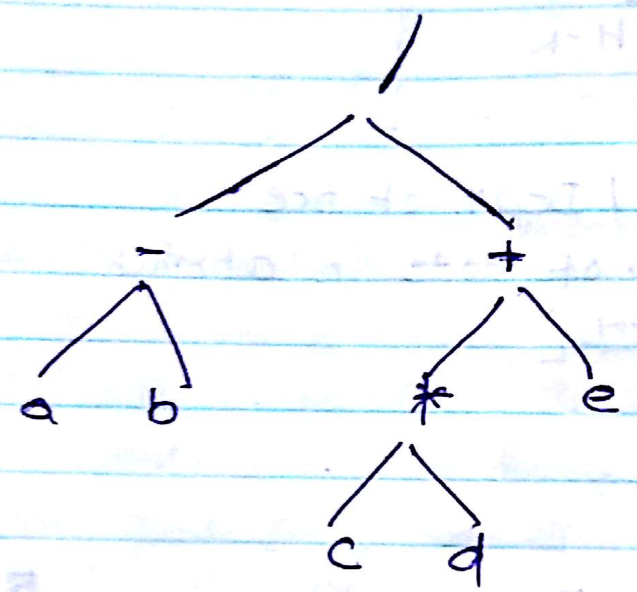
Copies a c

b and d are niether similar nor copy of each of each other.

* we judge the similarity of two trees on the basis of left sucessor and right Sucessor.

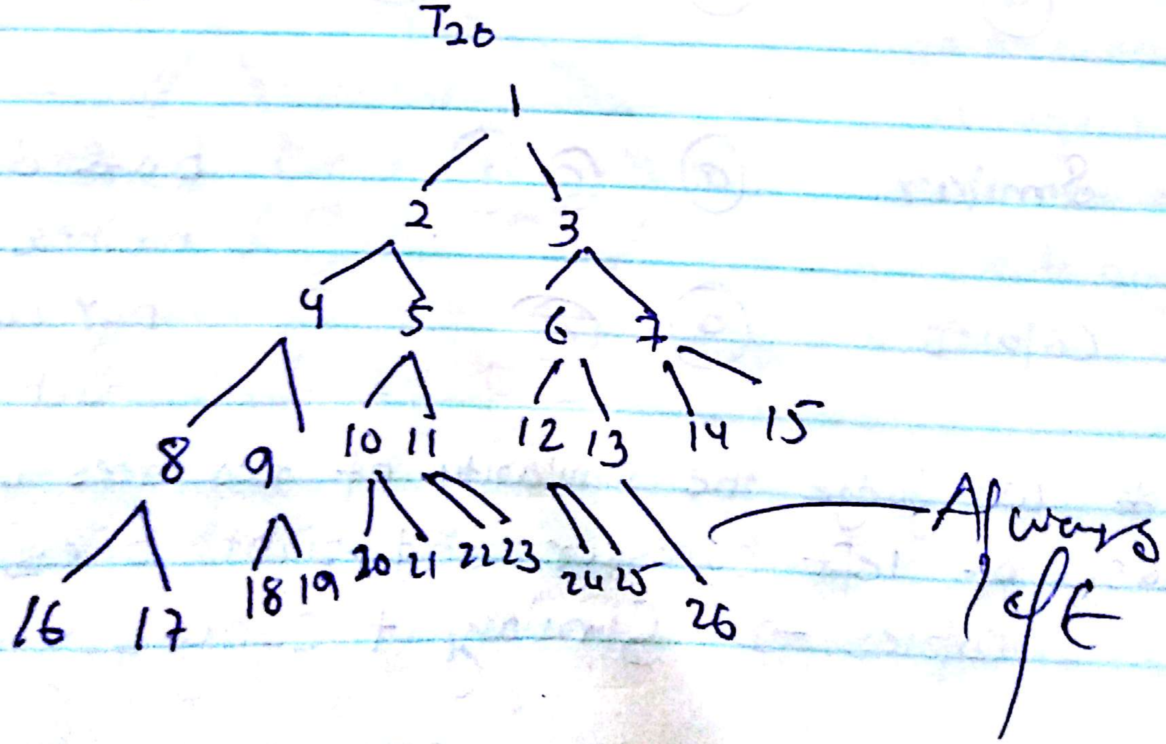
copies → Similary + copy →

$E = \underset{\text{I}}{(a-b)} \;/\; \underset{\text{II}}{((c*d) + e))}$



Complete binary trees -

It is said to be complete as it has max^m no. of possible of nodes except the last node.



For node K left child = $2*k$

Right child = $2*K+1$

~~present~~ parent = $\lfloor K/2 \rfloor$

e.g for node, $K=9$

left child = 18

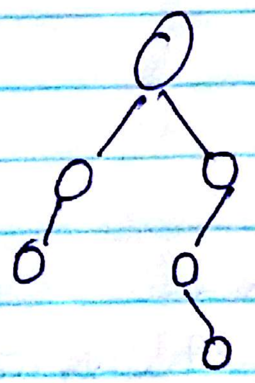
right = $18+1=19$

parent = $[9/2] = 4$

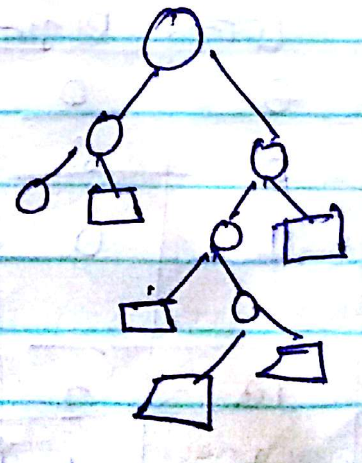
extended Binary tress → | 2-Tress

In this tree each node n has either zero or two children. node with two children called internal node. node with zero children called external node. We use circle for internal nodes and squares for external nodes

Binary tree



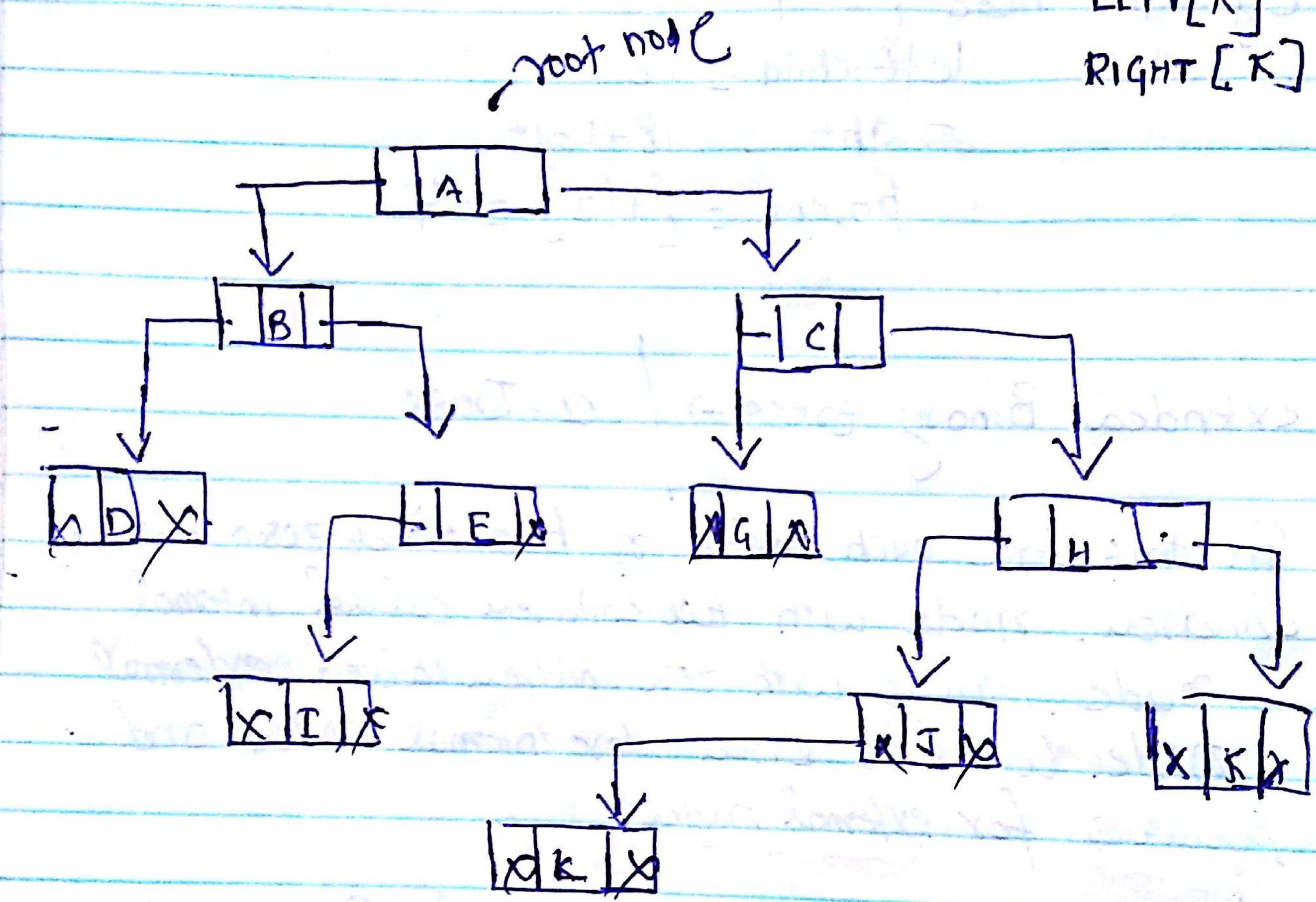
extended Binary tress



Representation of Binary tree in memory.

• Link representation

INFO [K]
LEFT [K]
RIGHT [K]



INFO	LEFT	RIGHT
K	0	0
C	3	6
G	0	0
A	10	2
H	17	1
L	0	0

B	18	13
D		
E	0	0
F	12	0
I	7	0
D	0	0

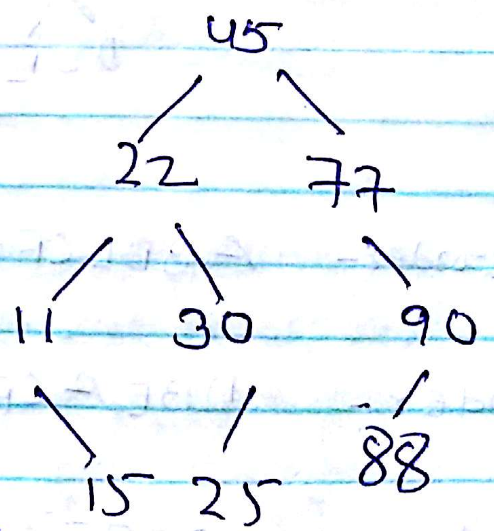
Sequencial representation - / Array

The root of tree is stored in

TREE
45 root
22
77
11
30
90
15
25
88

left → TREE[2K]

RIGHT → TREE [2K+1]



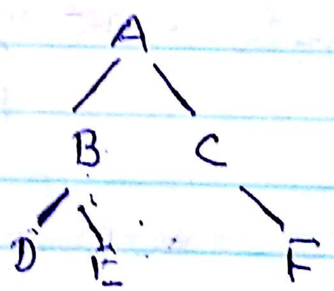
Sequence representation

Imp

Transvering in Binary tree -

*

Q



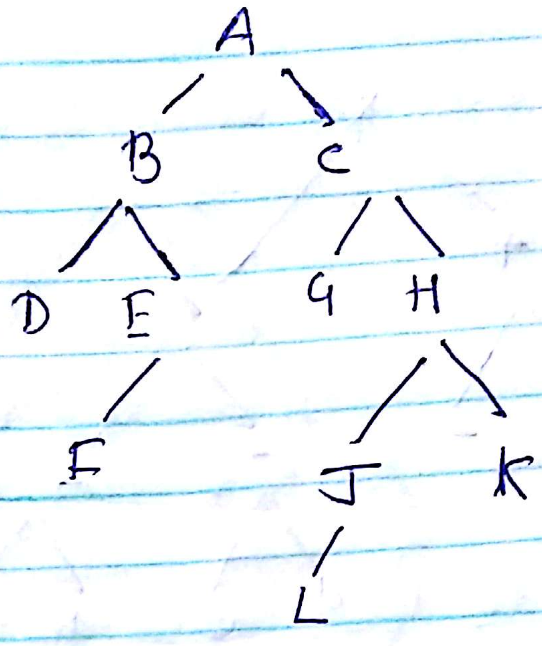
Pre-order- ABDECF

Inorder- DBEACF

Postorder - DEBFCA

Note -* Whenever the new node is counter apply the same order.

Q



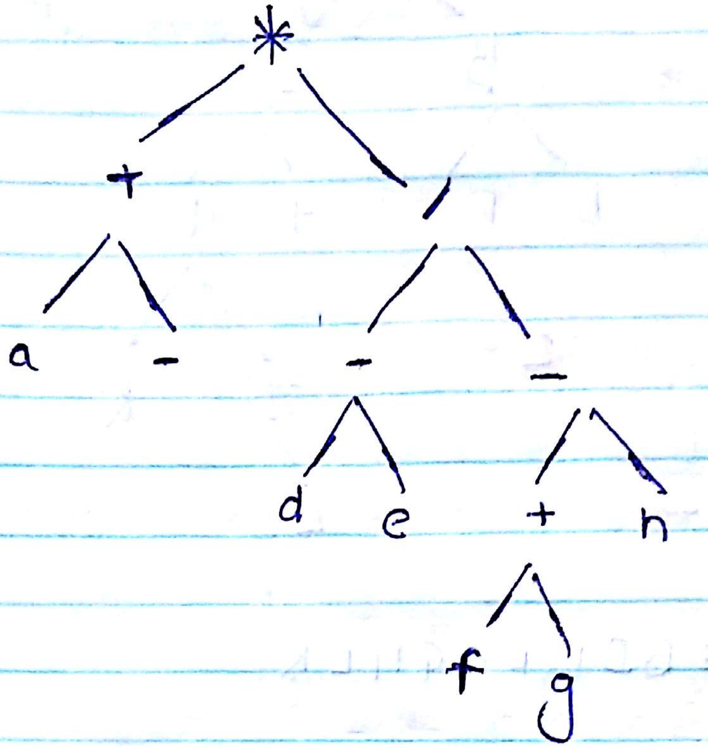
Preorder - A B D E F C G H L K
(root-left-right)

Inorder - D B F E A G C L J H K
(left-root-right)

Post-order - D F E B G L J K H C A
(left-right-root)

Q From the following given algebric expression, draw the binary tree then find out pre-order post order.
inorder

$$[a+(b-c)] * [(d-e)(f+g-h)]$$



(?)

Pre-order - *+a-bc/-de-+fgh

~~Inorder~~-

Postorder - abc-+de-fg+h-/*

V.Imp

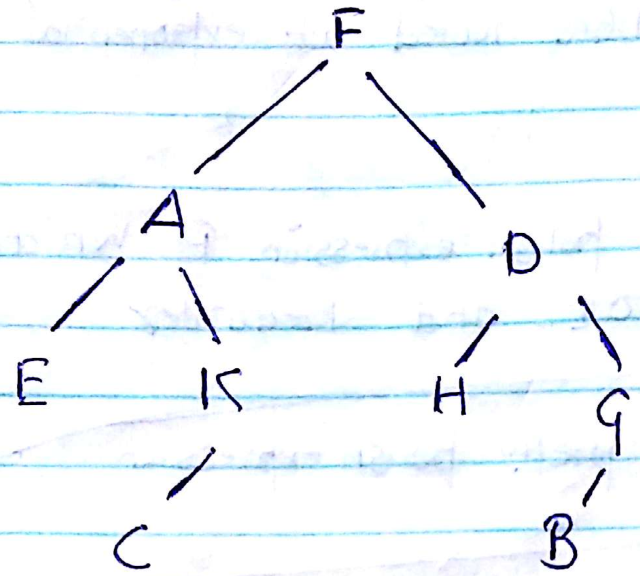
Q A binary tree P has 9 nodes, the inorder and preorder of tree yield the following sequence of node. Draw tree

left right

Inorder - E A C K F H D B G

Preorder - F A E K C D H G B

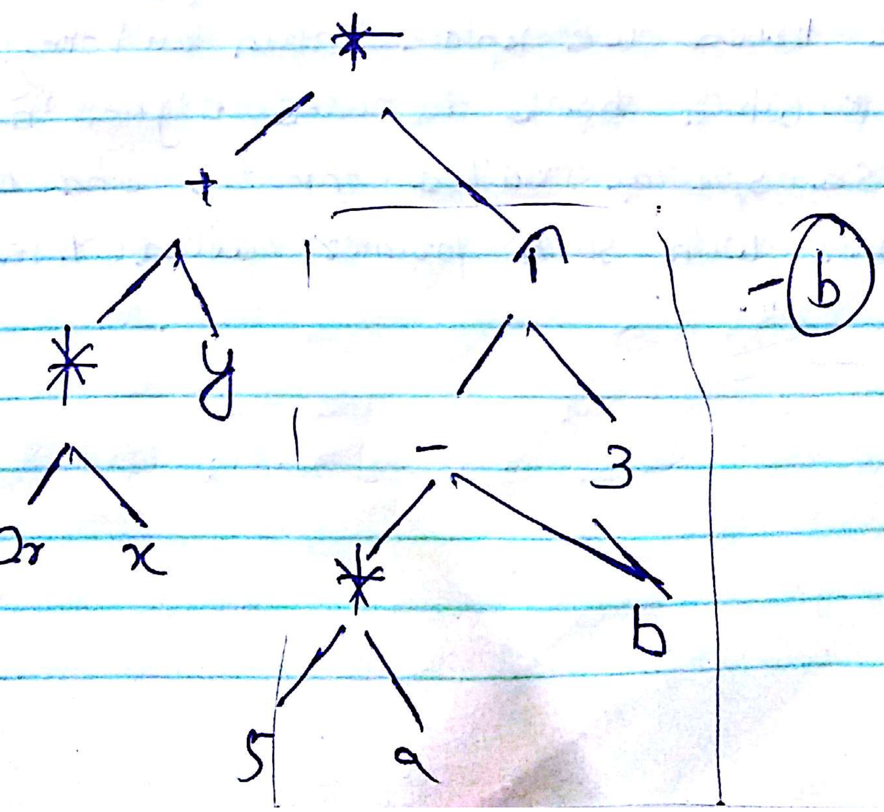
root



Q Consider the algebric expression

$$E = (2x+y)(5a-b)^3$$

(a) Draw the tree T which Corresponds to expression E.



(b) Find the Scope of exponential operator is find the subtree rooted at exponential

(c) Find the prefix polish expression P which is equivalent to E and Predorder

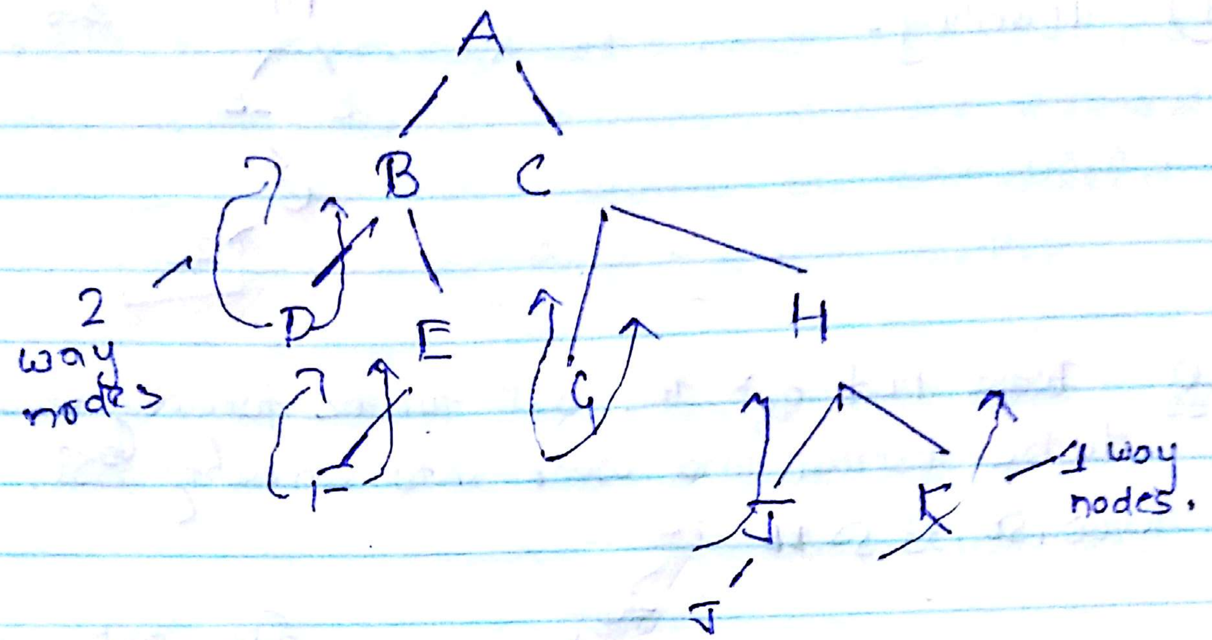
Preorder = prefix polish expression

$* + * 2 x y \uparrow - * 5 a b 3$

Threadings -

Some times we replace certain null entry by special pointer which points to nodes higher in nodes, These special called threading and binary trees with such pointer called threaded trees.

Null entries - those who don't have left entries.



V. Imp

Binary Search trees - (BST)

left - Smaller than Parent

Right - Larger than parent

↳ Search in BST

① Compare item with the root node. If item < n proceed to left child of node. If item > n proceed to right child of node.

② Repeat step-1 until following

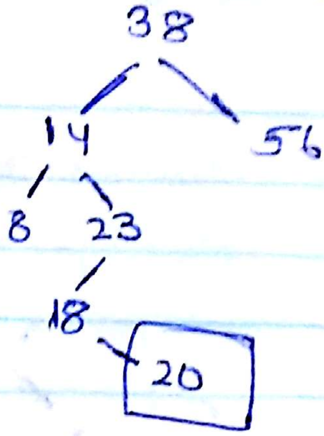
③ If item = n

Search Sucessfull

else

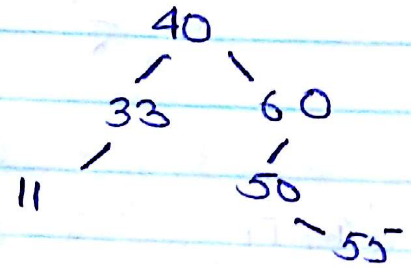
Search is Unsucessful.

Insert - 20



(ii) Inserting -

Q ~~Draw~~ Find out the BST for the following element to be inserted into order into an empty BST.
40, 60, 50, 33, 55, 11



Show each step in papers.

* During insertion, compare each item with the root node and proceed further.

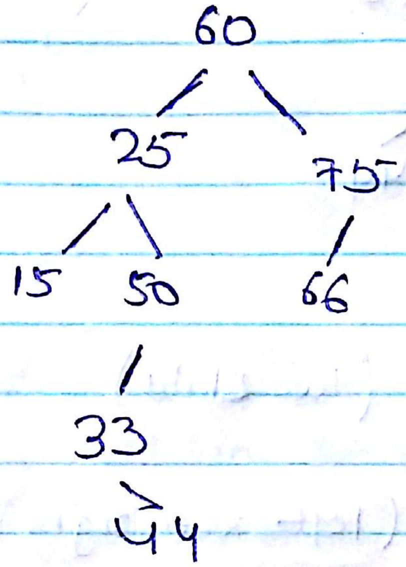
(iii) Deletion -

case - (i) N has children (leaf node)
N is deleted from the tress by simply replacing the nodes in the parent node by null pointer.

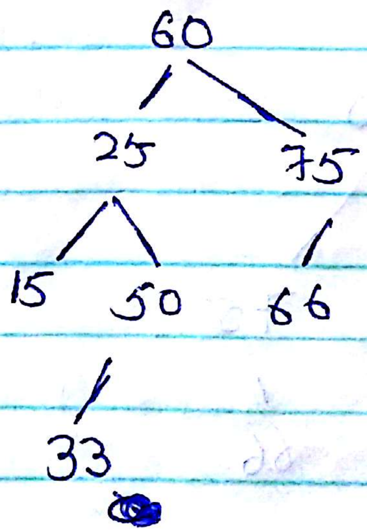
case - (ii) N has exactly one child.
N is deleted from the tree by simply replacing the location of n by parent nodes by the location only child nodes.

case (iii) N has two children

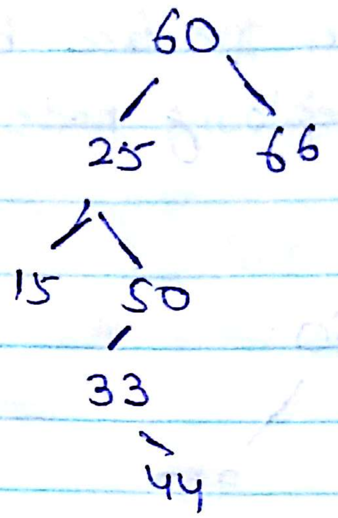
let S(N) denotes in order sucessor of Node N
N is deleted from the tree by first deleted S(N) and then replacing node N by the node S(N).



(1) Delete node 44 (no child)



Case-II Item = 75 (one child)

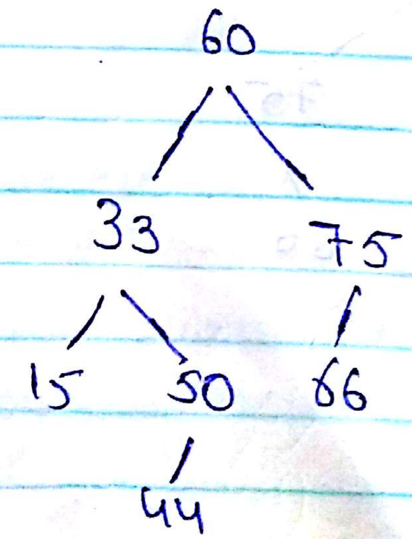


Case-(III) Item = 25 (two child)

find Inorder (left root right)

(V) ↑ replace

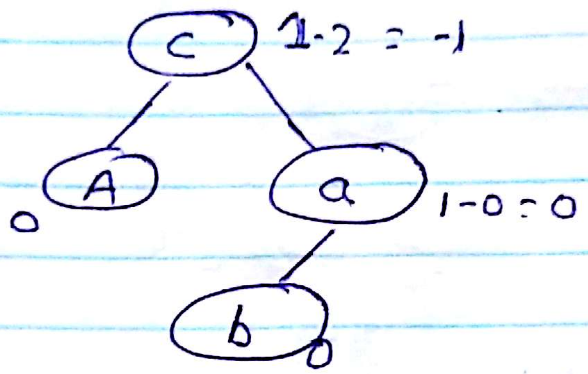
15 25 33 44 50 60



RL
represents the right subtree of node A

AVL tress →

$h^L - h^R$ ∈ 0, 1, -1



LL Rotation
Insert node is in left subtree of Node A

RR rotation
Insertion node is in the Right subtree of node A

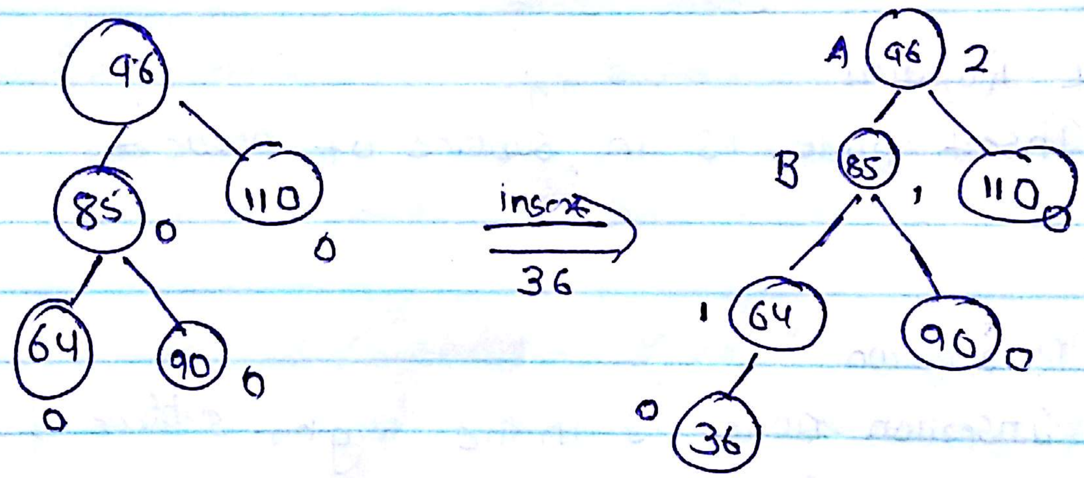
R-L

LR

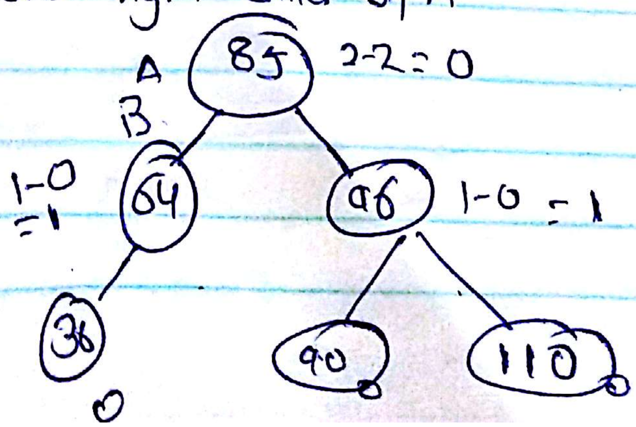
Inserted node in Right subtree

R-L notation

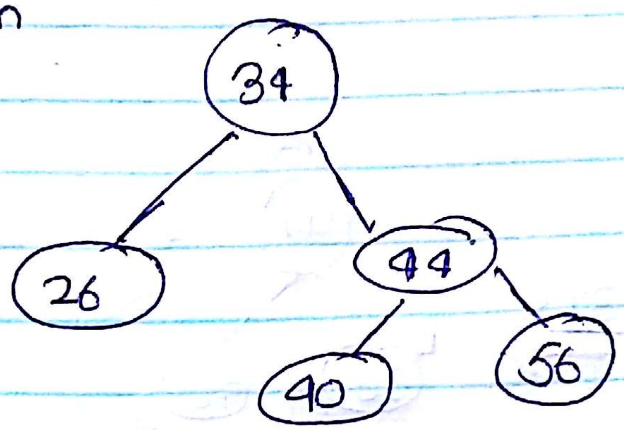
e.g. LL rotation



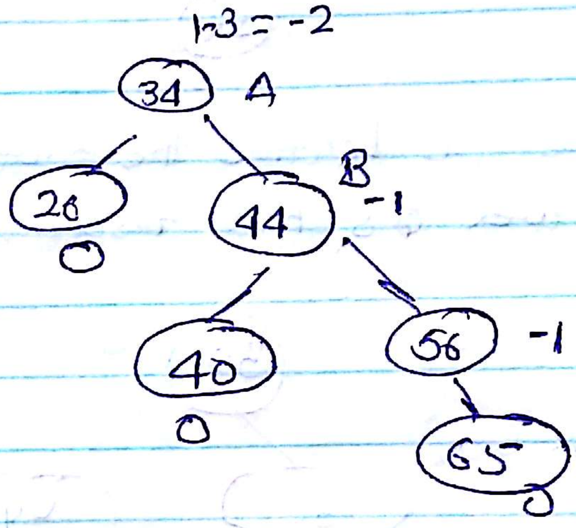
To balance it, ~~to~~ B becomes the root with B_L and A as its left and right child. B_R and A_R becomes left and right child of A



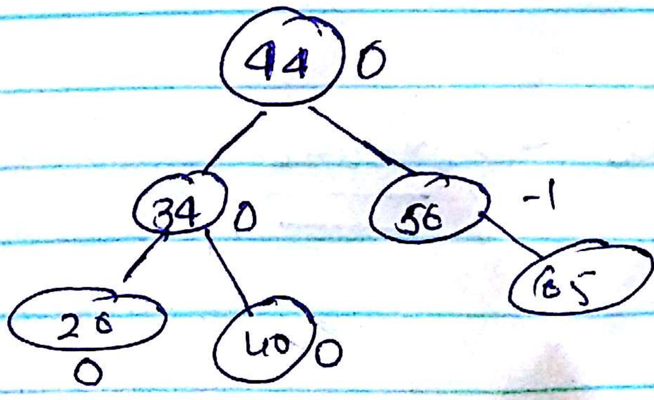
RR rotation



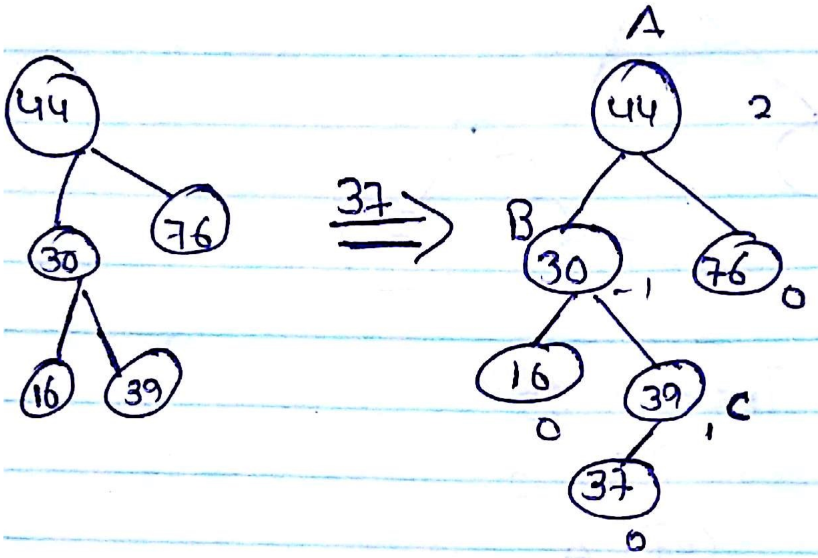
Insertion 65



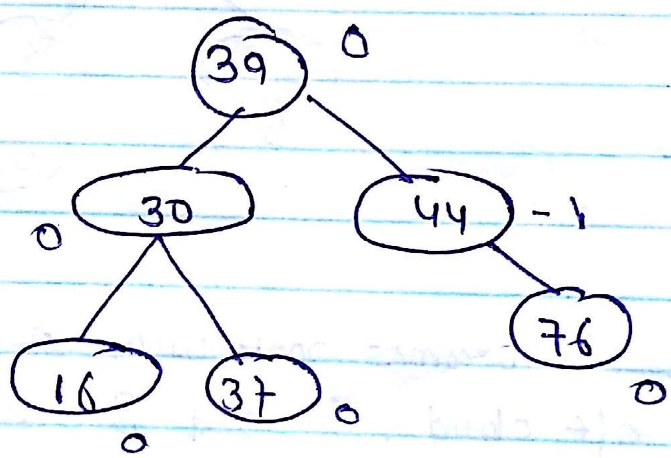
to balance, B becomes root with A and B_R as its left child, A_L and B_L becomes left and rt child of A



L-R



to balance, c becomes the root, B and A become left and r8 child respectively.



Date :

Page No :

each of the children of

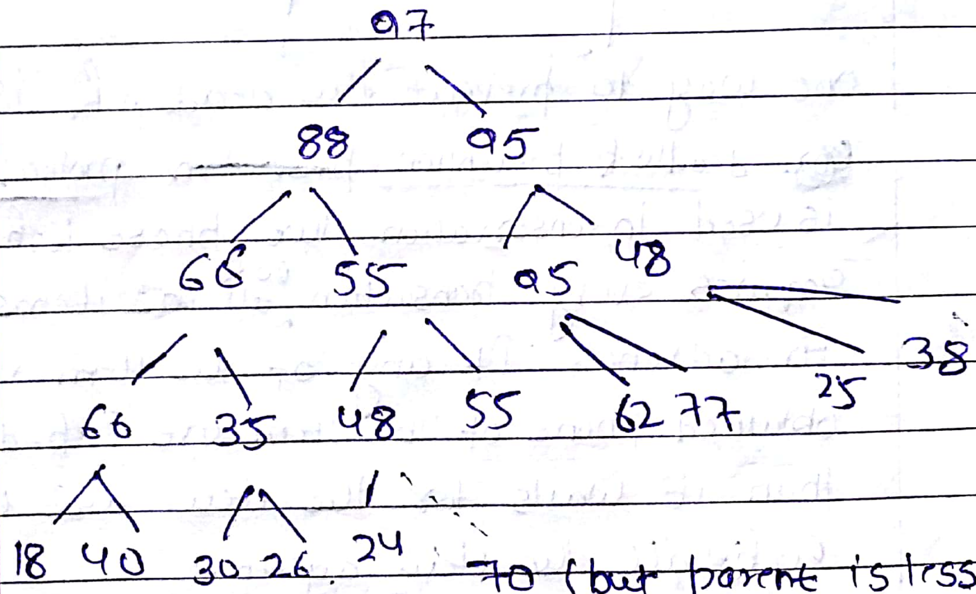
Heapsort - the value @ N is greater than ~~and~~ or equal to each children of N.

Note - ~~the value of N is less than N is~~ the value at N is less than or equal to the value at any children of N.

Insertion into heap -

(i) First add the item at the end of heap so that ~~so~~ heap a complete tree but not a heap,

(ii) Item is resume to its appropriate place so that H is finally a heap.



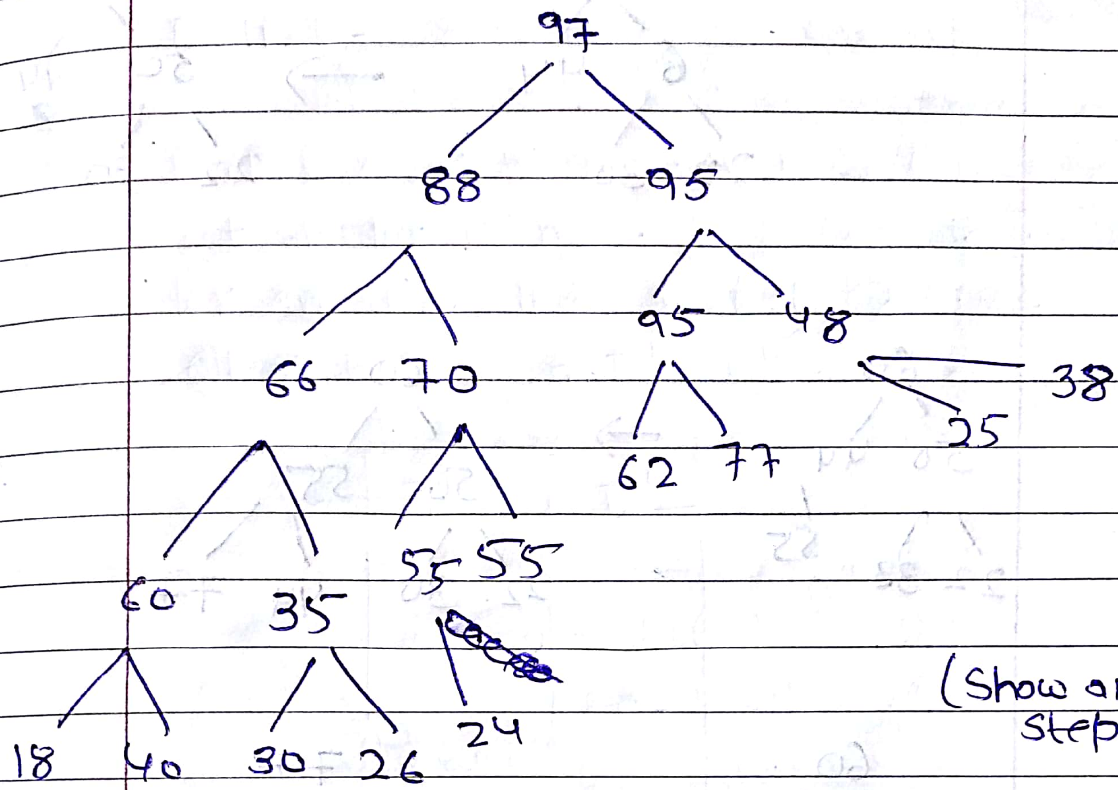
70 but parent is less and swap again and again.

Insert

* there is no small value in left (not compulsory)

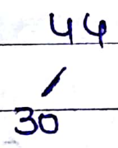
Date :

Page No :

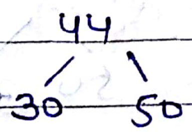


(Show all step).

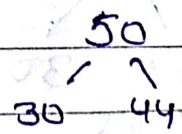
Q4 Build a heap from the following list of no.
44, 30, 50, 22, 60, 55, 77, 55

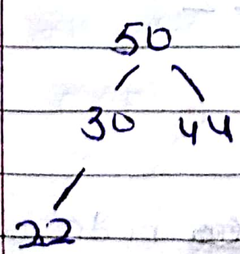


(check)



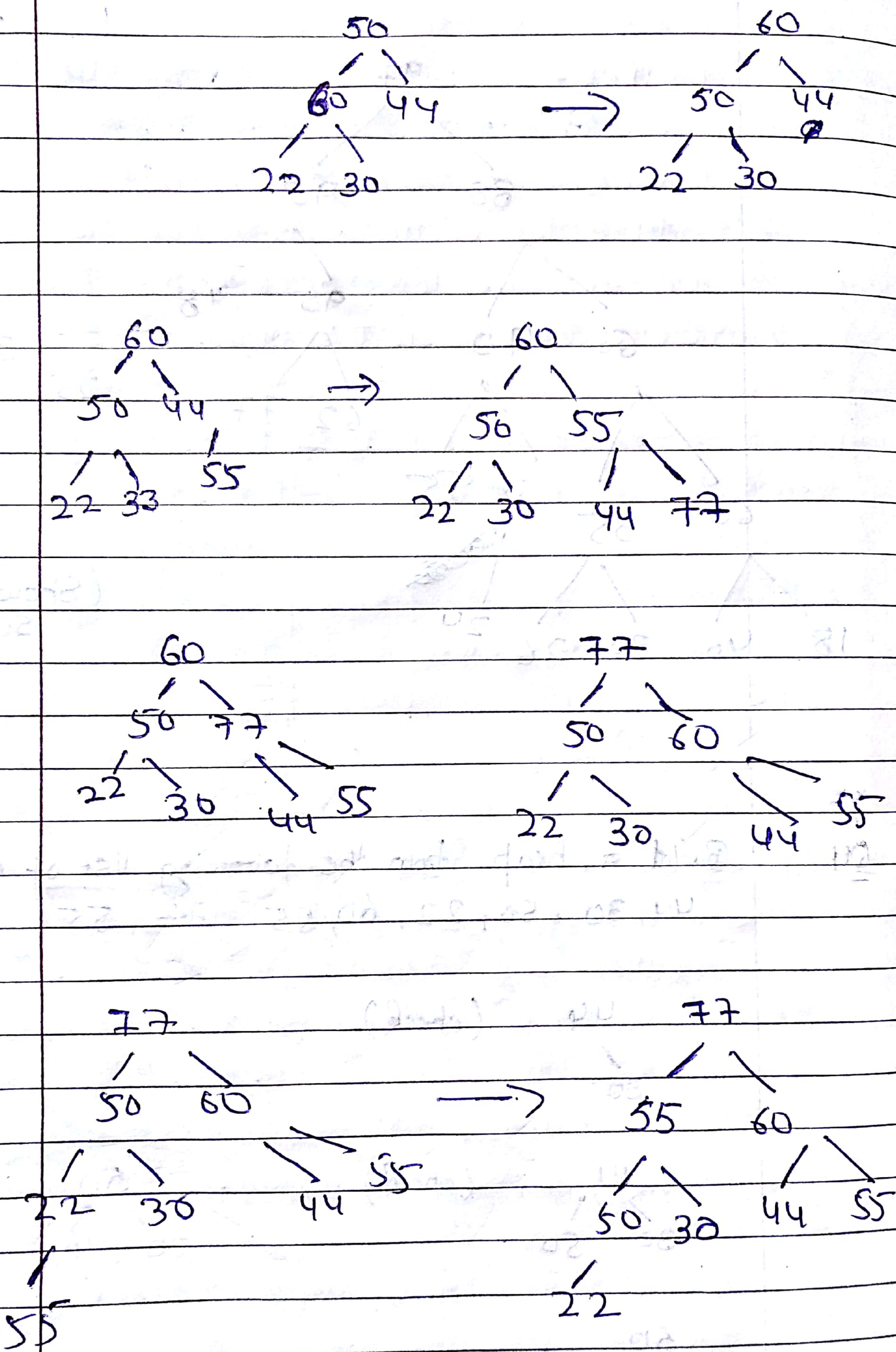
(check)





Date :

Page No :



Note - * Compare with the node at each step.

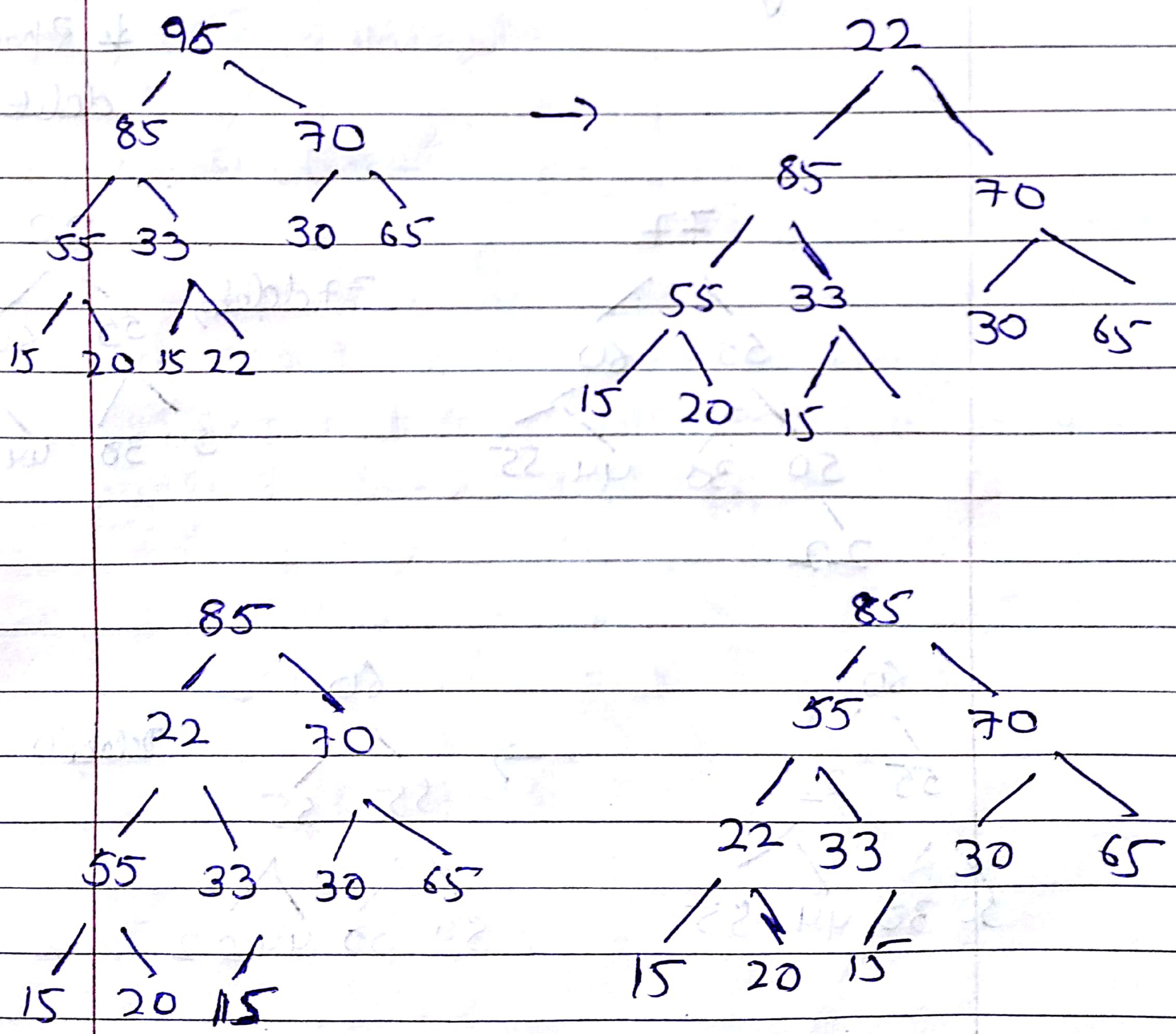
(Alway root node deleted)

⇒ Deleting the root of a heap.

1.) Assign the root to some variable item.

2.) Replace the deleted node R, by the last node by the L. of heap, so that H is a a complete tree but not necessary a heap.

e.g



Date :
Page No :

element obtained in descending order.

⟹ Heap sort algorithm

phase - A

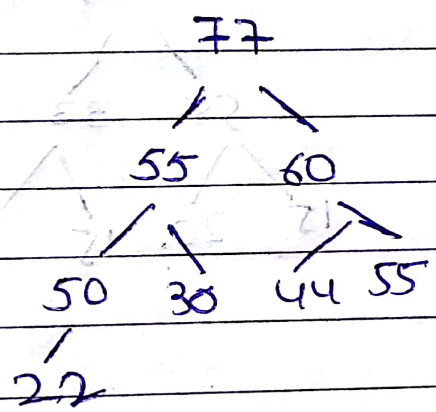
Build the heap out of the array or element A.

phase - B

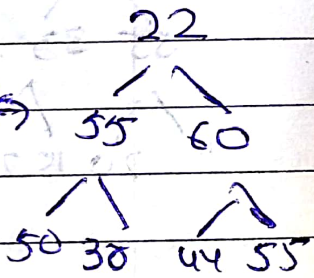
Repeatedly delete the root element of H, since the root of H always contain the large node in H.

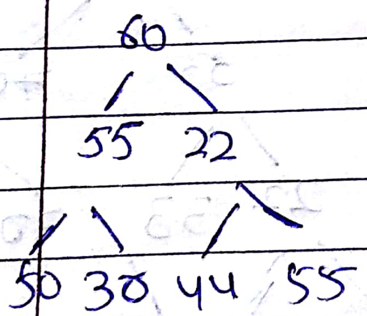
taking QH →

* Build heap
* Repeatedly delete root node

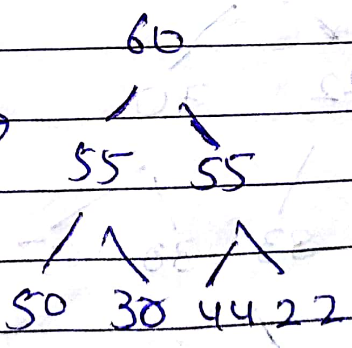


77 delete →

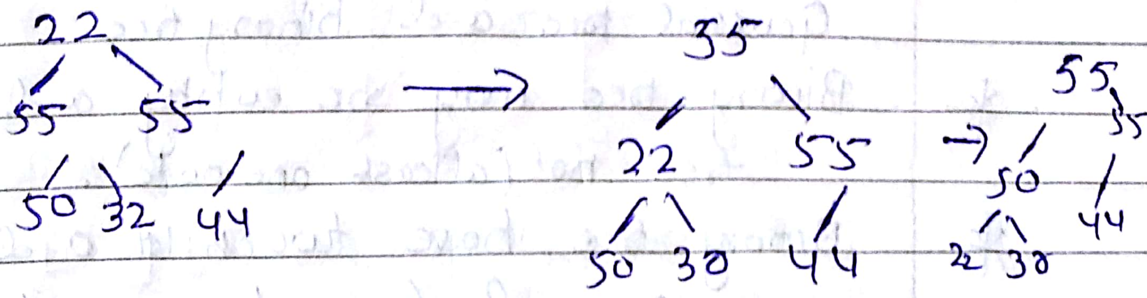




→

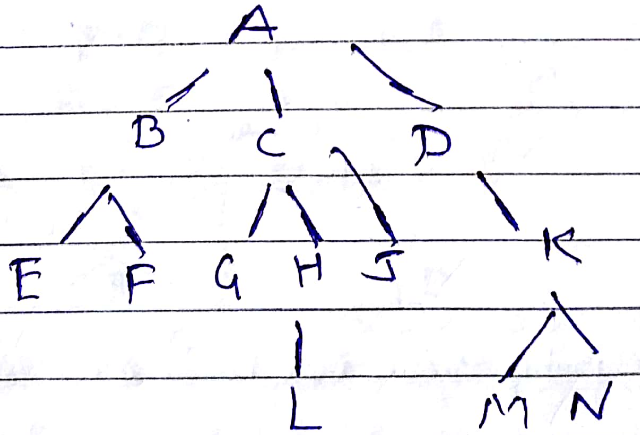


Delete 60



~~the~~ obtained element i.e deleted element is in descending order.

General tree -
It is a non-empty finite set of element called nodes such that P contain distinguish element R called the root. the remaining of T form an ordered collection of 0 or more disjoint tree $t_1, t_2 \dots t_n$



A is root node, root tree is alway unique
A, C have three children
B, K " 2 "
H, D " 1 child
E, F, G, L, J, M, N have no child (terminal or leaf nodes)

Date :
Page No :

General tree and binary tree

* Binary tree may be empty and General tree not (atleast one node).
* Binary tree have two child and but in general tree have N childreen.

*=) Memory representation of General tree-

(i) INFO[K]
(ii) Child [K] - the Ist of of K called of K.

(iii) SIBL [K] - other childred of K.

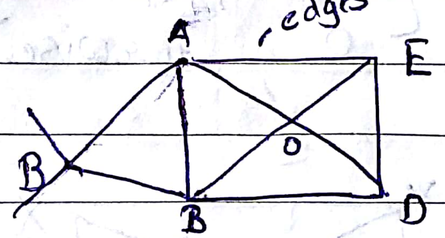
Graph

$G = (V, E)$

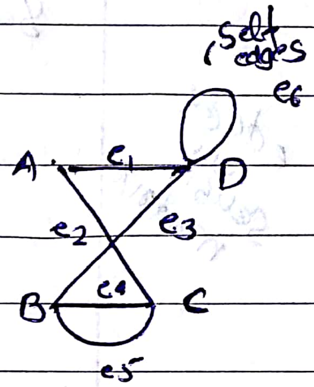
V = set of vertices / nodes

E = Edges

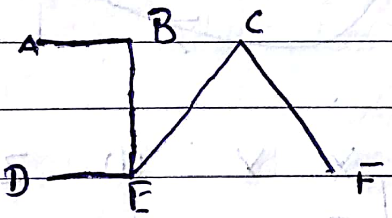
$e = [u, v]$



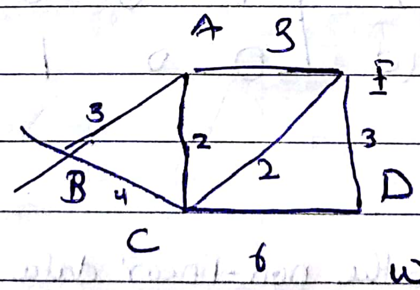
Graph



Multigraph



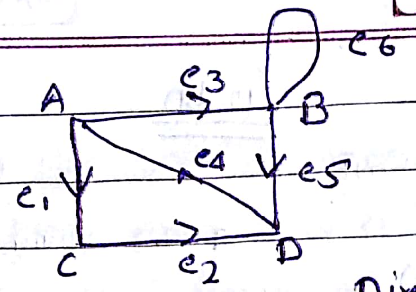
Tree



weighted graph

Date :

Page No :



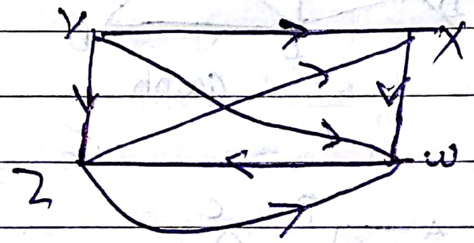
Directed graph -

e_2 [CD]

Sequencial representation of graph -

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacend to } v_j \\ 0 & \text{otherwise} \end{cases}$$

edges sensi matrix



adjacency matrix

$$A = (a_{ij}) = \begin{array}{c|cccc} & X & Y & W & Z \\ \hline X & 0 & 0 & 0 & 1 \\ Y & 1 & 0 & 1 & 1 \\ Z & 1 & 0 & 0 & 1 \\ W & 0 & 0 & 1 & 0 \end{array}$$

← bit matrix

Graph is the non-linear data structure in which cycle can be possible and formed a closed loop.

* A Set V of element called nodes,
* A Set E of edge such that each edge e in E represent the order pair of node in V, e [u,v].

Date :

Page No :

Path matrix →

$$P_{ij} = \begin{cases} 1 & \text{, if there any path from } V_i \text{ to } V_j \\ 0 & \text{, otherwise} \end{cases}$$

$$P_{ij} = \begin{array}{c} \\ X \\ Y \\ Z \\ W \end{array} \begin{array}{c} \begin{array}{cccc} X & Y & Z & W \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

Link representation of a graph →

Node , Next , Adj →

Next ↓ Pointer that points next node

Adj → Adjacent nodes.

V.V.V. Imp

traversing a graph-

STATUS 1 (Ready)

STATUS 2 (Waiting)

STATUS 3 (Processed)

BFS (Breadth First Search) -

(I) Intialised all nodes to ready States . Status-1

(II) Put the starting node A into queue and change its Status to wait , Status-2

(III) Repeat 4 and 5 until queue is empty.

(IV) Remove the front node and change the status to processed sate.

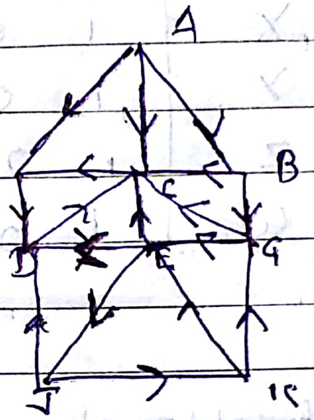
BFS is done to queue
DFS " " " Stack.

Date :
Page No :

(V) Add to the rear of queue all the neighbour of n that are in ready state and change their status to waiting state.

(VI) exit.

10 M



A - Starting
J - Destination
find shortest path A to J

Adjacent list

A :	F, C, B
B :	G, C
C :	F
D :	C
E :	D, C, J
F :	D
G :	C, E
J :	D, K
K :	E, G

Date :

Page No :

FRONT = 1 Queue = A

REAR = 1 ORIGINATOR : ϕ

→ A कहा से आया.

Remove front element A from the queue and add to queue to neighbour of A

F = 2 Queue : A, F, C, B

R = 4 ORIG : ϕ, A, A, A

Remove the front element F from the queue and add the neighbour of F

F = 3 Queue A, F, C, B, D

R = 5 ORIG : ϕ, A, A, A, F

Remove the front element C from the queue and add the neighbour of C

F = 4 Q : A, F, C, B, D (already in the Queue)

R = 5 OR : ϕ, A, A, A, F

Remove the front element.

F = 5 Que : A, F, C, B, D, G

R = 6 O : ϕ, A, A, A, F, B

Date :
Page No :

F=6 Q: A, F, C, B, D, G
R=6 O: φ, A, A, A, F, B

F=7 Q: A, F, C, B, D, G, E
R=7 O: φ, A, A, A, F, B, G

F=8 Q: A, F, C, B, D, G, E, J
R=8 O: φ, A, A, A, F, B, G, E

we stop now as J is added to queue is final destination.

we back track from J using originator

J ← E ← G ← B ← A

Date :
Page No :

Depth First Search.

* Initialised all nodes to ready State.
* Push the Starting node A onto the stack and change its to waiting state,
* Repeat Step 3 and 4 till Stack empty.
* pop the top node of Stack, process it and change its Status to process state.
* Push onto the stack all the neighbour that are still ready state and change its Status to waiting state.

exit.

Same e.g

Q. Find out all the nodes reachable to J including J itself.

Stack : J

Print J , STACK : D, K
Print K STACK : D, E, G
Print G STACK : D, E, C
Print C STACK : D, E, F
print F STACK, D, E,
Print E STACK D
print D STACK.

J, K, G, C, F, E, D

A Spanning tree is a tree that is derived from graph under the following condⁿ

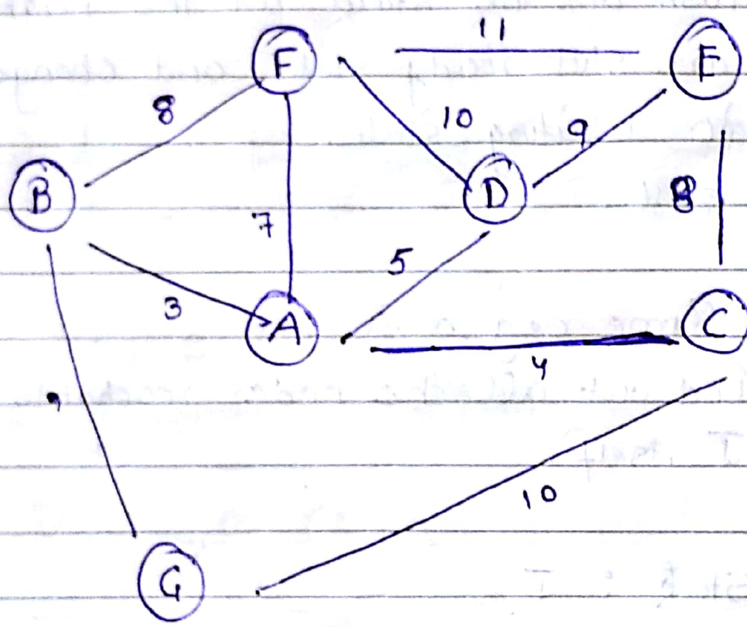
Date :
Page No :

* Weighted * Undirected
* Connected * have no loops or cycle

Minimal Spanning Tress. (NOT in Shaum Series)

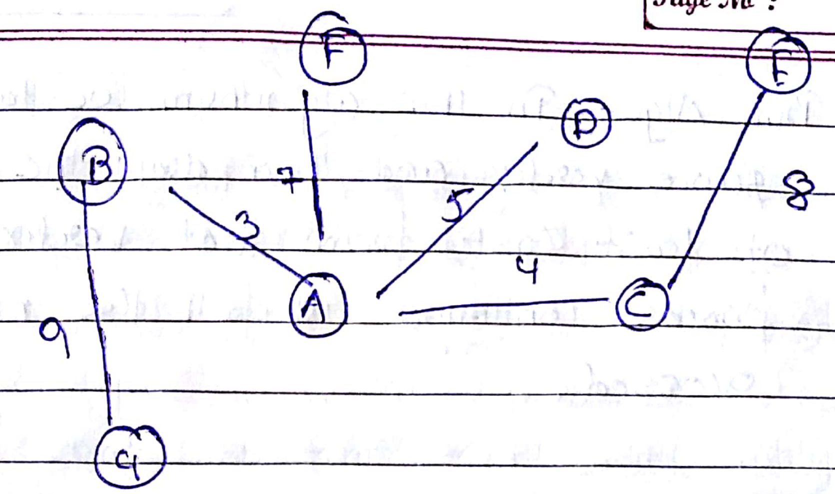
Spanning Tress having minimum cost called minimal Spanning tress and cannot have any cycle. * Network design

Krusal Algorithum-



* We Start with the edge of minimum weight and then chosse again the chose minimum cost from the edges of minimum vertex

Date :
Page No :

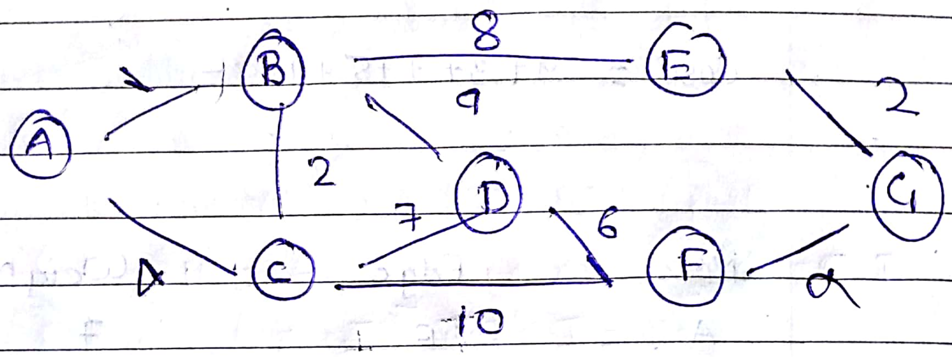


Cost = 9+3+7+5+4+8 =

Node	Edge	Weight	
A	AF	7	✓
	AC	4	✓
	AD	5	✓
B	BF	8	(x)
	BG	9	✓
C	CE	8	✓
	CG	10	x
D	DF	10	x
	DE	9	x
F	FE	11	x
E			
G			

Date :
Page No :

Prim Algo- In this algorithm we take a source vertex and then draw the edge of least cost from that vertex. This process continues till all the nodes are accessed.

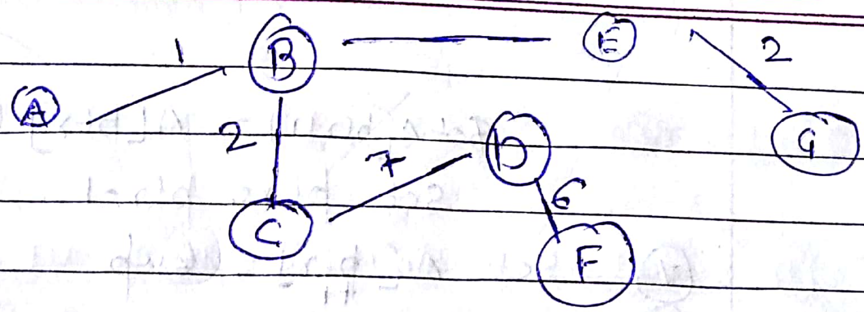


Let us assume A as Source vertex

Node	Edge	Weight	
A	AB	1	✓
	AC	4	X
B	BC	2	✓
	BD	9	x
	BE	8	✓
C	CD	7	✓
	CF	10	X
D	DF	6	✓
F	FG	9	x
E	EG	2	✓
G			

Date :

Page No :



Cost = 1+2+7+6+2+8 = 26 units

Insertion sort -

Sort the following element

77, 33, 44, 11, 88, 22, 66, 55

n-1 , imp

Pass	A[0]	A[1]	[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1	-∞	77	33	44	11	88	22	66	55
K=2		33	77	44	11	88	22	66	55
=3		33	44	77	11	88	22	66	55
=4		11	33	44	77	88	22	66	55
=5		11	33	44	77	88	22	66	55
=6		22	11	33	44	77	88	66	55
=7		22	11	33	44	66	77	88	55
=8		22	11	33	44	55	66	77	88

(I) Set A[0] = -∞ (sentinen)

(II) Repeat Step 3 to Step 5 K = 2, 3, ... N

(III) Set temp = A[K], ptr = K-1

(V) Repeate While (temp < A[ptr])

not pointer.

Set A[ptr+1] = A[ptr]

Set ptr = ptr-1

(V) Set A[ptr+1] = temp

(VI) exit.

Select Sort -

(II) Repeat for J = 0, J1, N ---- N-1

(1) Set MIN = A[J] AND loc = J

(2) Repeat for , I = J+1, J+2 ---- I<n

if MIN > A[I]

loc = i ;

Small = a[j] ;

loop 2

Interchange

temp = a[j] ;

a[j] = a[loc] ;

a[loc] = temp ;

loop1.