Date 14/10/22

Saathi

Data structures : Organisation of data (storage) in memory and perform operation on them.

Data store करना और उसपर operations करना.

⇒ Types :

1. Primitive and non primitive data ~~type~~ structures
2. Linear and non-Linear
3. Homogeneous & non-homogeneous.
4. Static and dynamic data structures.

1.1) Primitive D.S :- fundamental DS/ Basic DS
e.g int, float, char etc.

1.2) Non premitive : Combination of several premitive D.S e.g array, stack, linked list

2.1 Linear D.S : If data is sequential then data structure is linear e.g array, stack

2.2 Non-Linear D.S :- e.g Tree, Graph

3.1 Homogeneous D.S : All elements in D.S are of same type called homo. DS

```
struct xyz
{
    int no;
    int age;
};
```

```
struct abc
{
    int no;
    float height;
};
```

non-homogeneous 3.2

Date ___/___/___

4.1 Static & dynamic

Static D.S allocates in Compile time.

4.2 dynamic D.S allocates in run time.

Types of D.S tree

- Primitive (Basic/fundamental)
  - int
  - float
  - Char
  - pointer
- Non-Primitive (Complex)
  - Linear (sequential)
    - Array
    - List
    - Stack
    - Queue
    - LinkedList
  - Non-linear (Non-sequential)
    - Tree
    - Graph
  - Files

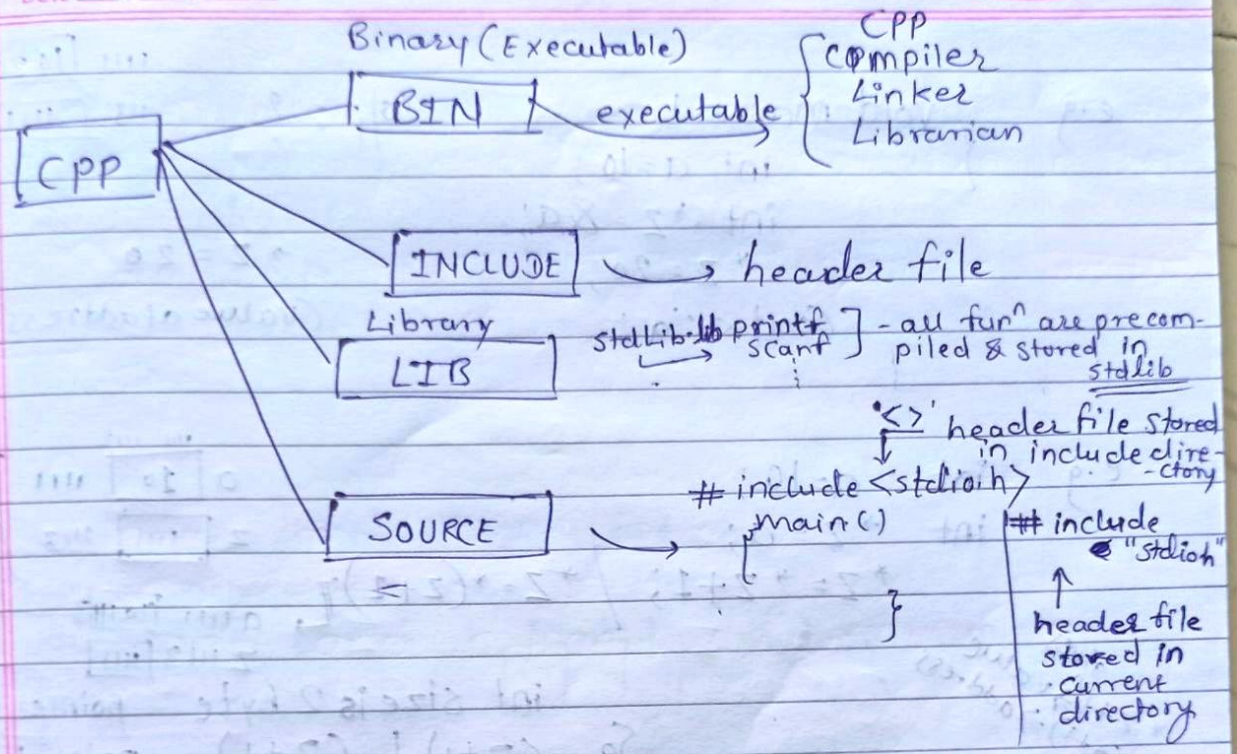
Q. IDE ?

SCCS
Source code control System

- Editor
- Preproccessors
- Compiler
- Linker
- SCCS
- librarian

IDE provides all of this.

Date ___/___/_____



Linker : Links our obj code with Library functions from the libraries by actually making the library fun^n's as a part of our code

⇒ Pointers.

All type of pointers get the same number of byte to store an address irrespective of what they are pointing to.

& = " Adress of "
* = " Value at address "

++, *

Saathi

Date ___/___/___

e.g
```
void main()
{
    int a=10;
    int *z=&a;
    *z=20;
    Cout<<a;
}
```

o/p : ?

1111 [20] a
1113 [1111] z

*z = 20

(value at address 1111) = 20

e.g
```
int a=10;
int *z=a;
*z=*z+1;
```
/ *z=*(z+1)

a [10] 1111 } initial
z [1111] 1113

a 1111 [1113]
z 1113 [1111]

*z+1 ⇒ value
*(z+1) ⇒ address
gives

int size is 2 byte

So, (z+1) → 1112 | (z+1) → 1113

pointer int ला point करतोय म्हणून z ची size ही 2 byte ने वाढेल

e.g
```
#include<stdio.h>
#include<conio.h>
void main()
{
    Char a='B';
    char *z=&a;
    ClrscR();
    printf("&a=%x\n", &a);
    printf("z=%x\n", z);
    *z++;
    printf("(*z) z=%x\n", z);
    printf("%x", a);
    return;
}
```

o/p: &a = fff5
z = fff5
(*z++) z = fff6

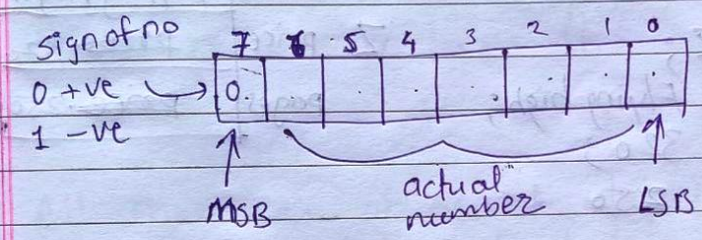
zero is positive number.

Date ___/___/_____

```
void main()
{
    char []a={'A', 'B', 'C'};
    char *z=a;
    printf("%c %c", a[0], z[0]);
```

o/p = A A

* Signed & unsigned (MSB/LSB)



1 Byte can store +128 to -128 number means 0 to ±127

e.g
```
void main()
{
    int c ; clrscr();
    c = 400*400/400;
    printf ("%d", c);
}
```

o/p : ?

⇒ Dynamic memory allocation.

C++ → New, delete.
↑ are operators.

→ is pointer operator

Saathi

Date ___/___/___

⇒ Structure in Cpp :

```
*  Struct Book                    or    Class Book
   {                                    {
       Char name;                          //
       int price;
       int pages;                          }
   };
```

Memory

	name	Flying high
Z	price	~~price~~ 320
	pages	~~price~~ 250

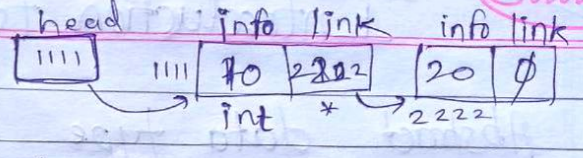
```
node Z;
Z.name = Flying high;
Z.price = 320;
Z.pages = 250
```

or

```
node *z;
z = new node;
z->name = Flying high;
z->price = 320;
z->pages = 250;
```

```
*  Struct node
   {
       int info;
       struct node * link;
   };

   Struct node a;
   Struct node b;
```

Page No.




```
Struct node * head;
    head = &a;
    a.info = 10;
    a.link = &b;
    b.info = 20;
    b.link = ϕ;
```

① Diff in struct & class

Both are similar the diff is -

Struct	Class
1) All members of struct are public by defaut	1) By default members of class are private.

16-10-22
lecture 4

Date / /

Introduction to DS

☆ Abstract data type

It is a mathematical model of DS.
e.g TV remote, fan speed regulator.

It does not implement, it just provide way to implement.

☆ Algorithmic notations.

⇒ properties of Algorithm

1. Name of Algorithm.
2. Introductory Comment.
3. Steps
4. Comments - include only for clarity & performs no actions
5. Statement & Control Structure
   a. Assignment statement (←)
   b. Symbol (=) use as relation operator
   c. If statement
   d. Case statement.
   e. Repeat statement.
6. Go To and Exit loop Statements.
7. Exit ~~No~~ statement
8. Variable Names. (use in capital letter)

Date ___/___/___

⇒ Data structures

(1) Array ⇒ ARRAY [DIM1, DIM2 ...]
name subscript

(2) Dynamic storage

⇐ use for allocation

Q. Write an algorithm to find largest of two numbers

Algorithm LARGEST. This algorithm read two numbers and finds the largest of these two numbers

Step 1] [Read the first Number]
Read [A]

Step 2] [Read the second No.]
Read (B)

Step 3] [Compare A&B & store the largest no in C]
If A > B
then
C ← A
else
C ← B

Step 4) [print value of C]
write (C)

Step 5) Finished
EXIT

lecture 6

Date ___/___/___

⇒ Format conventions,

1) Arithmetic operations & expressions

Exponential (↑) Solve from Right to left.

e.g 2↑3↑2 i.e 2^{3^2} = 1024

2) Relations & Relational operator.

<, >, <=, >=

3) Logical operations & Expression.

!, &&, ||

4) Input and output.

5) Sub algorithm

a) function - Single value return.

b) procedures - No value return.

The purpose of Sub algorithm is to perform Some Computation when required under Control of algorithm

e.g

```
void main ()
{
    int a=10;
    int &b=a;
    printf("%d", b);
}
```

o/p = 10

a, b → [10]

here b is reference to a

e.g india is reference by hindustan or bharat

Page No.

lecture 7
17/10/22

Date ___/___/___

Saathi

★ Time and space analysis

Asymptotic Analysis — program which will take less time & space.

Nature of curve when data size increases this study is known as Asymptotic Analysis

★ Calculate time complexity

1) [Initialize value of SUM]
   SUM ← 0 — 1
2) [calculate the sum] — n+1
   Repeat For I = 1, 2, ... N
   SUM ← SUM + A[I] — n
3) [Print the sum]
   Write [SUM] — 1
4) [Finished]
   Exit — 1

$1+n+1+n+1+1 = 2n+1$

Time Complexity $= O(n)$.

O (Big O) : Upper Bound (Worst case)
Ω (Big Omega) : Asymptotic Tigh Bound [Avg case]

Θ (Theta) : Lower Bound (Best case)

★ Accessibility of public, private, protected through various inheritance.

public	public	public, X, protected
private	Private	protected, X, protected
protected	Protected	private, X, private

Page No.

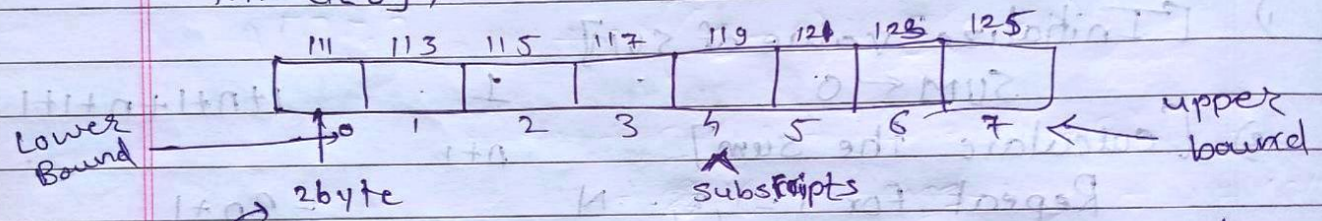
18-10-22
lecture 9

Date __/__/____

Unit II
Array.

1) An array can be defined as an infinite collection of homogeneous elements (similar type) ~~em~~ elements.
2) Array store multiple values which can be accessed by a single (same) name.
3) Arrays are always stored in consecutive memory locations.

int a[8];

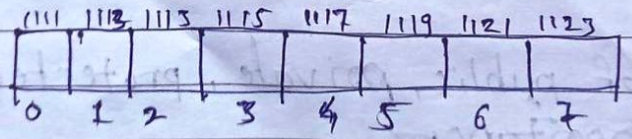


word size: Space occupied by one element of the element

★ Types
1) One dimensional
2) Multi dimensional.

⇒ Storage representation

A[I] = Base_address + Word_Size * (I - Lower_Bound)



Base Add = 1111
word size = 2 bytes
Lower Bound = 0

A[6] = 1111 + 2 * (6 - 0)
= 1111 + 2 * 6
= 1111 + 12
= 1123

Date ___/___/_____

Saathi

Two dimensional array can be stored in following two form.

1. Row major form
2. Coloumn major form.

C/C++

	0	1	2
0	10 (1111)	20 (1113)	30 (1115)
1	40 (1117)	50 (1119)	60 (1121)
2	70 (1123)	80 (1125)	90 (1127)

3x3

Row major form

1111	1113							1127
10	20	30	40	50	60	70	80	90

other programming languages

Coloumn major

10	40	70	20	50	80	30	60	90

⇒ Accessing elements of arrays

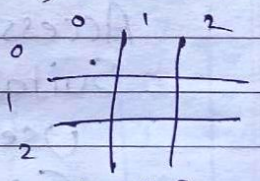
1) Row Major form

$$A[I][J] = BA + WS * (N * (I - LB_r) + (J - LB_c))$$

2) Coloumn Major

$$A[I][J] = BA + WS * ((I - LB_r) + M * (J - LB_c))$$

where,
- i - row
- j - coloumn
- BA - Base Add
- WS - word size
- M - no of rows
- N - no of coloumn
- LB_r - Low Bound of row
- LB_c - lower Bound of coloumn.



A[0][0]

↑ LB_r ↑ LB_c

Page No.

e.g Row major form

$$A[i][j] = BA + WS * (N * (i - LB_r) + (j - LB_c))$$

	0	1	2
0	11	90	32
1	71	63	24
2	24	88	84

11	90	32	71	63	24	24 88	84 88	84

1125

$$A[2][1] = 1111 + 2(3(2-0) + (1-0))$$
$$= 1111 + 2(6+1) = 1111 + 14$$
$$= 1125$$

e.g Coloum major

$$A[i][j] = BA + WS * ((i - LB_r) + M * (j - LB_c))$$

$$A[1][2] = 1111 + 2 * ((1-0) + 3(2-0))$$

$$= \cancel{1111 + 2(2+6)} = 1111 + 2 * (1+6)$$
$$= \cancel{1111 + 16} = \cancel{1127} = 1111 + 14 = 1125$$

1111	1113	1115	1117	1119	1121	1123	1125	1127
11	71	24	90	63	88	32	24	84

⇒ Advantage

1) Access any element randomly
2) Suitable when no of elements in Advance
3) Used in designing other data structure
4) Consist similar kind of element

⇒ Disadvantages

1) We have to specify the size of array using compile time
2) Wastage of memory
3) Cannot change size in runtime.

Date ___/___/_____

⇒ Applications in

1) used Matrix operation.
2) Used to sort elements.
3) Polynomial addition and substraction.

Polynomial in x with degree 4.

exponents

$2x^4 + 2x^3 + 5x^2 + 15x^1 + 3x^0$

coefficient

$2x^5 + 2x^3 + 2$

$2x^4 + 2x^3 + 2x + 1$

Polynomial Addition

poly1

i

2	5
2	3
2	0

j

2	4
2	3
2	1
1	0

k=

2	5
2	4
4	3
2	1
3	0

Page No.

lecture 10

Date ___/___/___

⇒ **Stack**

1) Array representation
2) Link representation

It is linear collection of data items where insertion, deletion take place on one end called top of Stack.

~~prg~~ ~~# include <iostream>~~ push function
~~using~~ ~~stdname~~

```
# define MAX 5
int top = -1;
int stack[MAX];
void push (int item)
{
    if( top == MAX-1)
    {
        printf (" stack is full");
        return;
    }
    top = top+1;
    Stack[Top] = item;
    return;
}
```

* pop function

```
int ~~void~~ pop ()
{
    int item;
    if (top == -1)
    {
        print (" Stack is empty");
```

Date ___/___/_____

```
        return -1;
    }
    item = stack[top];
    ~~return~~ top = top - 1;
        return item;
}
```

★ peek()

```
int peek()
{
    int item;
    if (top == -1)
    {
        print(" Stack empty");
        return -1;
    }
    item = Stack[top];
    return item;
}
```

★ display()

```
{
    int i = top;
    if (top == -1)
    {
        print (" Stack empty");
        return 0;
    }
    for (i = top; i >= 0; i--)
    {
        print ("%d", stack[i]);
        return;
}
```

Page No.

Date ___/___/___

⇒ infix to prefix & postfix

A*(B+C)*D	A*+BC*D
A*BC+*D	*A+BC*D
ABC+**D	**A+BCD
ABC+*D*	

α

2*3+(4+2)*3+10	2*3+42*3+10
2*3+(42+)*3+10	*23+42*3+10
23*42+*3+10+	**23+423+10
23*42+3*+10+	++**23+42310

562+*124/-

2 6 5	8 5	6+2	4 12 40	5*8	3 40	3 12/4

40-3

37

Date ___/___/_____

Operator	Priority
#	0
(	1
+, -	2
*, /	3

prog infix to postfix.

```
# include <iostream>
# include <conio.h>

# define MAX 50;
  int stack[MAX];
  int top=-1;

void push (char elem)
{
      top++;
      stack[top]= elem;
      return;
}

~~int~~ char pop()
{
      char elem;
      elem= stack[top];
      top--;
      return elem;
}
```

Page No.

Date ___/___/_____

```
char peek()
{       char item;
        item = stack[top];
        return item;
}
int pr(char ch)
{ Switch ~~pr~~ (~~char~~ ch)
  {
        case '#' : return 0;
        case '(' : return 1;
        case '+' :
        case '-' : return 2;
        case '*' :
        case '/' : return 3;
  }
}

void main()
{
        int infx[50], pofx[50], ch, elem;
        int i=0, j=0;
        cout << " Enter infix expression";
        cin << infx;
        push('#');

while ((ch = infx[i++]) != '\0')
{       if (ch == '(')
            push(ch);
        else
            if(isalnum(ch))
                pofx[j++] = ch;
```

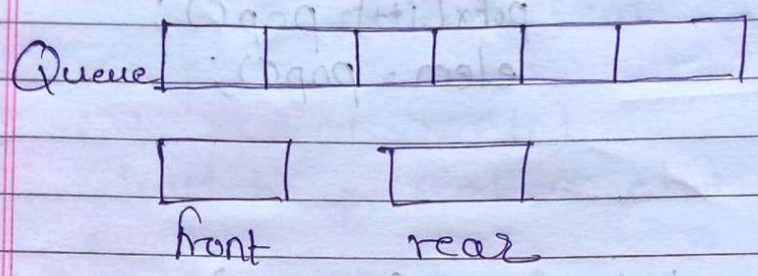
```
else
    if (ch==')')
    {
        while (peek()!='(')
            pofx[j++]=pop();
            elem=pop();
    }
else
    {
        while (pr(peek()) >=pr(ch))
            pofx[j++]=pop();
            push(ch);
    }
}
while (peek()!= '#')
{
    pofx[j++] = pop();
    pofx[j] = '\0';

cout<<" pofx expression is"<<pofx;
}
```

19/10/20

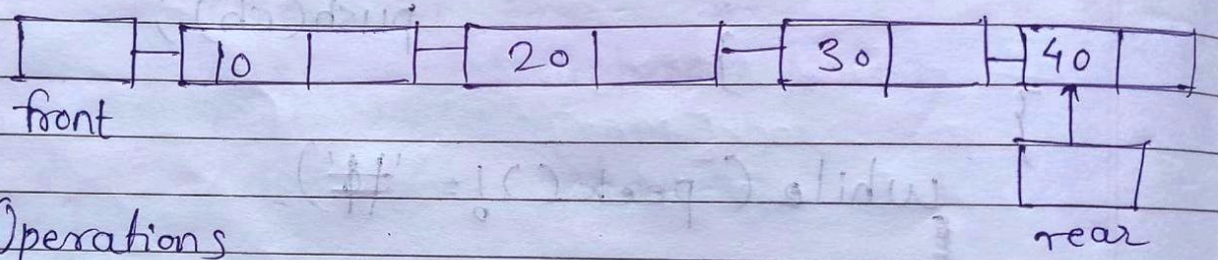
Date ___/___/___

⇒ Queue :-

Representation of queue :- ① using array.



② using linked Representation



⇒ Operations

1)
```
void enqueue(int x)
{
    if (rear == MAX-1)
    {
        cout<<" ~~Link list~~ Queue is full";
        return;
    }
    if (front == -1)
        front = 0;
    rear++;
    queue[rear] = x;
    return;
}
```

Date ___/___/___

Saathi

2)
```
int dequeue()
{
    int x;
    if (front == -1 & rear == -1)
    {
        cout<<" Queue is empty\n";
        return -1;
    }
    x = queue[front];
    if (front == rear];
        front = rear = -1;
    else
        front ++;
        return x;
}
```

⇒ Types of queue

1) Linear queue
2) Circular queue
3) Double ended queue
   1. input restricted dequeue
   2. Output restricted dequeue
4) Priority queues.

⇒ Circular queue

```
void enq(item){
    if (front == 0 && rear == MAX-1) || (front == rear+1))
```

Date ___/___/___

```
{       cout<<" Queue Overflow\n";
            return;
        }
        if (front == -1)
        {
            front = 0;
            rear = 0;
        }
    else
        {
            if (rear == MAX-1)
                rear = 0;
            else
                rear++;
        }
            queue[rear] = item;
    }
```

2) Dequeue :-

```
int dequeue()
{
    int item;
    if (front == -1)
    {
        cout<<" Queue underflow";
            return -1;
    }
        item = queue[front];
    if (front == rear)
    {   front = rear = -1; }
```

```
else
{       if (front == MAX-1)
            front = 0;
        else
            front++;
    }   return item;
}
```

3) ① Display

```
void display()
{
    int i;
    if (front == -1)
    {
        cout<<" Queue is empty\n";
            return;
    }
    cout<<" Queue elements:\n";
        if (front <= rear)
        { for (i=front; i<=rear; i++)
            cout<<" cqueue[i]<<"\n";  }
    else
    {
        for (i=front; i<= MAX-1; i++)
            cout<< cqueue[i];
        for (i=0; i<=rear; i++)
            cout<< cqueue[i];
    }
        cout<<"\n");
            return;
}
```

Date ___/___/___

⇒ Double ended queue

1) we can add elements front both ends & delete element from both ends

(*) input restricted
Add into the double ended queue at rear, dequeue at both ends.

(*) output restricted double ended queue
Add at both end, delete only at front.

⇒ Priority queue

Elements are deleted (dequeue) based on some priority

lecture 13
19/10/22

⇒ Linked Representation of stack and queue.

Array	Linked List.
1) Static Data structure	1) Dynamic data structure
2) Size cannot be change at runtime	2) Size can be changed at runtime.
3) If less memory has define then program may fall short of memory	3) Efficient management of memory

Date ___/___/_____

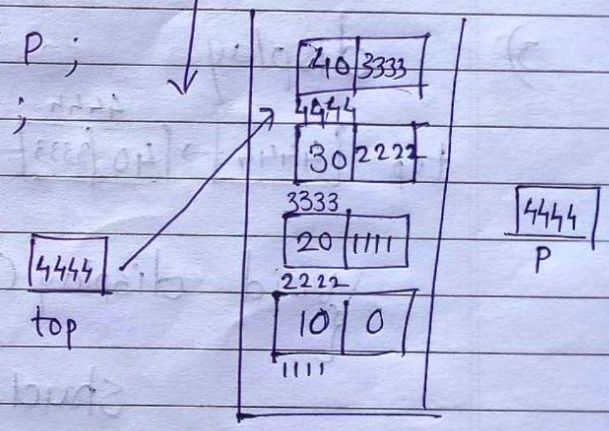
Saathi

* Stack using Linked list :

1) push

```
~~void push()~~ struct node
{
        int info;
        struct node *link;
    };
    int top = 0;

void push(int x)
{
        struct node *p;
        p = new node;
        p→info = x;
        p→link = 0;
    if (top==0)
        top=p;
else
    {
        p→link = top;
        top = p;
    }
    return;
}
```

top to Bottom processing



2) pop

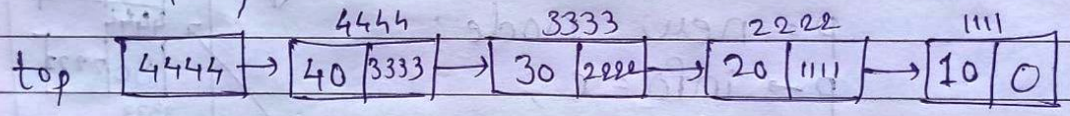
```
int pop()
{
    int x
    struct node *p;
```

Page No.

Date ___/___/_____

```
if (top==0)
{
        cout<<"stack is empty";
            return -1;
    }
    x= top->info;
    p=top;
    top=top->link;
     delete p;
     return(x);
}
```

3) display

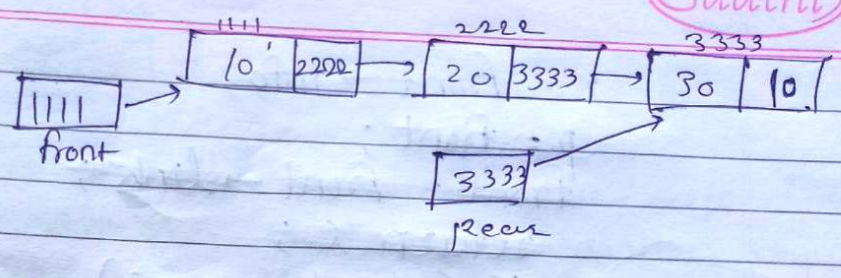



```
void display()
{
        struct node *p;
            p=top;
        if(p==0)
        {
            cout<<"Stack empty";
            return -1; }
while (p!='0')
{
        cout << p->info;
        p = p->link;
    }
        return;
}
```

Date ___/___/___

Saathi

⇒ Queue



1) Enqueue :

```
void enqueue (int x)
{
        struct node *p;
        p = new node;
        p → info = x;
        p → link = 0;

    if (front == ϕ && rear == ϕ)
            front = rear = p;
    else
        {
            rear → link = p;
                rear = p;
        }
            return;
    }
```

2) Dequeue :

```
Int dequeue()
{
        int x;
        struct node *p;
    if (front == ϕ && rear == ϕ)

    {   cout<<"Queueue is empty";
            return -1;
    }
```

Page No.

```
x = front -> info;
p = front;
~~front = front -> link;~~
~~return x;~~
if ( front == rear)
    front = rear = 0;
else
        front = front -> link;

    delete p;
    return x;
}
```

3)

```
void display()
{
        Struct node *t;
        t = front;
        if (t == q)
        { cout<<" Queue empty";
            return;
        }
    while (t != '\0')
{
        cout<<t -> info;
        t = t -> link;
    }
        return;
}
```

↑ behaviour is same as display in stack.

Lecture 14
20-10-2

Date __/__/__

Unit 3

Linked List.

It is a linear data structure. They are dynamic in nature allocates the memory when required linked list reduces the access time

Disadvantage

1. The memory is wasted as pointers require extra memory for storage.
2. No element can accessed randomly
3. Reverse traversing is difficult in linked list.

⇒ Applications

1. Use to implement stack, queue, graph.
2. we dont need to know the size in advance
3. Allows you to insert elements at begining

→ Types of Linked List

1. Singly
2. Circular
3. Doubly

Page No.

Date ___/___/_____

```
struct node
{
    int info;
    struct node *link;
};
```

⇒ Insert At Begin :

```
void insert_at_Begin(int x)
{
    Struct node *p;
    p = new node;
    p->info = x;
    p->link = 0;
    if (head==0)
        head=p;
    else
    {
        p->link=head;
        head = p;
    }
    return;
}
```

⇒ Insert At End

```
void insert_at_end (int x)
{
    Struct node *p, *t;
    p= new node;
    p-> info = x;
    p-> link = 0;
```

```
if (head==0)
    head=p;
else
{
    t=head;
    while (t->link!=0)
        t=t->link;
    t->link=p;
}
return;
}
```

Tricks

① while(t->link!=0)
t will contain Address of last node

② while (t!=0)
t=t->link
t will visit upto last node.

③ while (p->info!=20)
{ q=p;
p=p->link;
}
p, q contain address of current & previous node

⇒ Insert At Position.

```
void insert_at_pos(int pos, int x)
{
    struct node *p, *q, *t;
    int count=0;
    p= new node;
    p -> info = x
    p -> link = 0;
    If (head==0)
    {
        cout<<" Linked List empty";
        delete p;
        return;
    }
    if (pos==1)
    {
        p->link=head;
        head=p;
    }
```

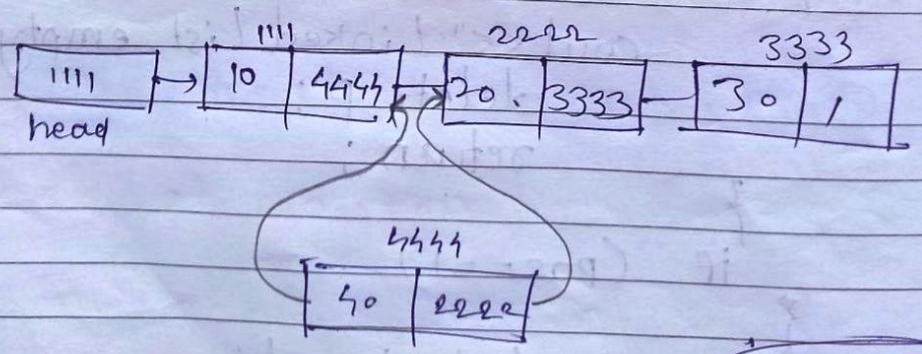
← If head is zero, then we need free memory allocated to new node p

Date ___/___/___

```
else
{
    t=head;
    Count=1;
while(Count!=pos && t!=0)
{
    q=t;
    t=t->link;
    Count++;
}
  if(t!=0)
{
    ~~q->link=pt;~~  q->link=p;
    ~~t->link=t;~~   p->link=t;
}
else
{
    cout<<"Invalid value of position\n";
delete p; }
}
    return;
}
```

If t becomes 0, means the position is not found
i.e if we exit this loop & t!=0 means we are at correct position.

उथुन बाहेर निघले तर position is found



pos = 2

lecture-15
22-10-22

Date ___/___/___

Saathi

⇒ ① Delete At Begin()

```
int delete_at_begin()
{
        struct *node p;
        int x;
    if (head==0)
    {
        cout<<"LL is empty";
        return -1;
    }
        p=head;
        x = head->info;
        head = head->link;
        delete(p);
        return x;
}
```

⇒ ② Delete At End

```
int delete_at_end()
{
        struct *node *t, *q;
        int x;
    if (head==0)
    {
        cout<<"LL is empty";
        return -1;
    }
        t=head;
        while (t->link!='\0')
{
        t=q;
```

Page No.

Date ___/___/___

```
            q->link=0;
            t = t->link;
q->link=0;  x = t->info;
            delete(t);
            return x;
        }

    if ( head->link == '\0')
    {
            t = head;
            x = head->info;
            head = 0;
            delete(t);
            return x;
    }
    t = head;
    while (t->link != '\0')
    {
            q = t;
            t = t->link;
    }
            q->link = 0;
            x = t->info;
            delete(t);
            return x;
}
```

⇒ Delete at position

```
int delete_at_pos(int pos)
{
        struct *node *q, *t;
```

Date ___/___/_____

```
int x, count;
if (head=0)
{
    cout<<"LL is empty";
    return -1;
}
if (pos==1)
{
    x= head->info;
    t= head;
    head= head->link;
    delete(t);
    return x;
}
```

delete at begin
जर head affect
होतो म्हणुन हे
block seperate
लिहलाय.

```
t=head; count=1;
while( count!=pos && t!=0)
{
    q=t;
    t=t->link;
    count++;
}
if (t!=0)
{
    x= t->info;
    q->link= t->link;
    delete(t);
    return x;
}
else
{   cout<<"Invalid position";
    return -1;
}
```

Page No.

Date ___/___/___

```
}
```

⇒ Display

```
void display()
{
    struct *node t;
        t=head;
    if (head==0)
    {
        cout<<"LL is empty";
        return;
    }
    while (t!=0)
    {
        cout<<t->info;
        t=t->link;
    }
    return;
}
```

← good programing practise to write return

⇒ Count

```
int count()
{
    Struct *node *t = head;
        int c=0;
        while ( t!=0)
        {
            t=t->link;
```

```
        c++;
    }
    return c;
}
```

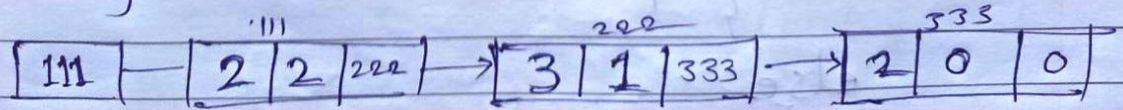
⇒ Search (int x)

```
{
    struct *node t;
        t=head;
    int found=0;
    While (t!=0)
    {
        if (t->info==x)
        {
            found=1;
            break;
        }
        t=t->link;
    }
    return found;
}
```

Q how to define polynomial expression using link list structure

Syntax → | Coeff | expo | Link |

e.g $2x^2 + 3x + 2$;



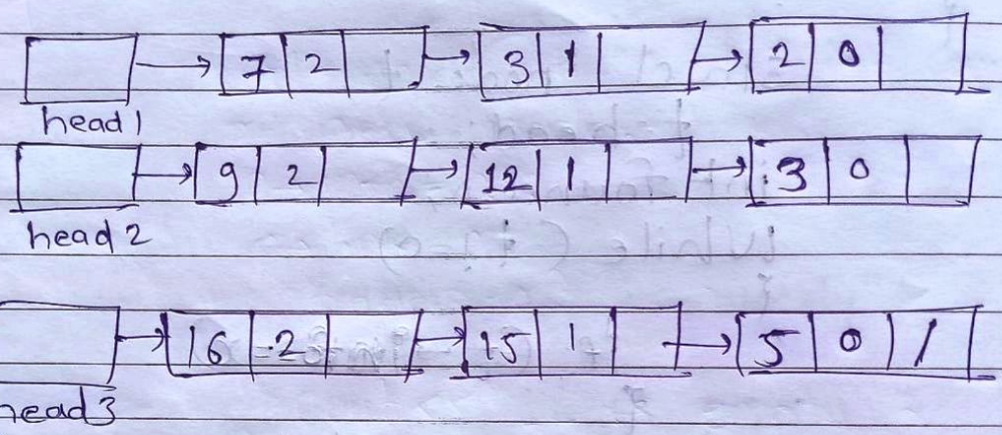
H.W

Q. how to store polynomial expression in link list & perform diff operation

e.g Concider the following two polynomial

1) $7x^2+3x+2$

2) $9x^2+12x+3$



lecture-16

★ Circular link list :-

```
void insert-at-begin (int x)
{
        struct node *p, *t;
        p= new node;
        p→info = x;
        p→link = 0;
        if (head==0)
        {
            head = p;
            p→link = p;
        }
    else
        {
```

```
else
{
        t = head;
        while (t -> link != head)
                t = t -> link;
                t -> link = p;
                p -> link = head;
                head = p;
}
        return;
}
```

⇒ Insert At End

```
void insert_at_end( int x)
{
        struct node *p, *t ;
        p = new node;
        p -> info = x;
        p -> link = 0;
        if (head == 0)
        {
                p -> link = p;
                head = p;
        }
else
{       t = head;
        while (t -> link != head)
        {
                t = t -> link;  }
                t -> link = p;
                p -> link = head;
}
```

Date ___/___/_____

```
    return;
}
```

⇒ Insert At position

```
void insert_at_pos(int pos, int x)
{
    Struct *node *p, *q, *t; int count=0;
      p = new node;
      p->info = x;
      p->link = 0;
    if (head==0)
    {
      cout<<"LL is empty";
        return;
    }
    if (pos==1)
    {
        t=head;
      while (t->link != head)
          t=t->link;
          t->link=p;
          p->link = head;
          head = p;
    }
  else
    {
        t= head;
        count;
      while ( count != pos)
```

Date ___/___/___

Saathi

```
{
        q=t;
        t=t->link;
        count++;
        if (t==head)
            break;
    }
    if (t!=head)
    {
        q->link=p;
        p->lint= t;
    }
else
        cout<<" Invalid position";

        return;
}
```

lecture A

⇒ Delete At ~~Position~~ Begin

```
int delete_at_~~end~~begin()
{
        struct node *p, *t;
        int x;
    if (head==0)
    {
        cout<<" LL is empty";
        return;
    }
        p= head;
        x= head->info;
```

```
if (head->link == head)
        head = 0;
else
{
        t = head;
    while (t->link != head)
        t = t->link;
    t->link = head->link;
        head = head->link;
}
        delete (p);
        return x;
}
```

⇒ Delete At End

```
int delete_at_end()
{ struct *t, *q;
    int x;
if(head == 0)
{
        cout << "LL is empty";
        return -1;
}
        if (head->link == head)
        {
            t = head;
            x = head->info;
            head = 0
        }
    else
        {
```

Date ___/___/_____

Saathi

```
        t=head;
        while(t->link != head)
        {
            q=t;
            t=t->link;
        }
        q->link=t->link;
        x=t->info;
    }
    delete(t);
    return x;
}
```

⇒ Delete At Position:

```
int delete_at_pos(int pos)
{
    struct node *p,*q,*t;
    int x,count;
    if (head==0)
    {
        cout<<"LL is empty";
        return -1;
    }
    if (pos==1)
    {
        x=head->info;
        p=head;
        if (head->link==head)
            head=0;
```

```
else
{
        t= head;
    while (t->link != head)
            t = t->link;
    t->link = head->link;
        head = head->link;
    }
        delete (p);
        return x;
}
    t=head;
    count=1;
while( count != pos)
{
        q=t;
        t=t->link;
        count++;
        if (t==head) break;
}
    if (t != head)
{
        x=t->info;
        q->link = t->link;
        delete t;
        return x;
}
    else
        {   cout<<" Invalid position";
            return -1;
        }
}
```

To check for invalid position

Date ___/___/___

Saathi

⇒ Display

```
void display()
{
    Struct node *t=head;
    if (head==0)
    {
        cout<<"LL is empty";
            return;
    }
    do
    {
        cout<<t->info;
        t=t->link;
    } while(t!=head);
    return;
}
```

⇒ Count

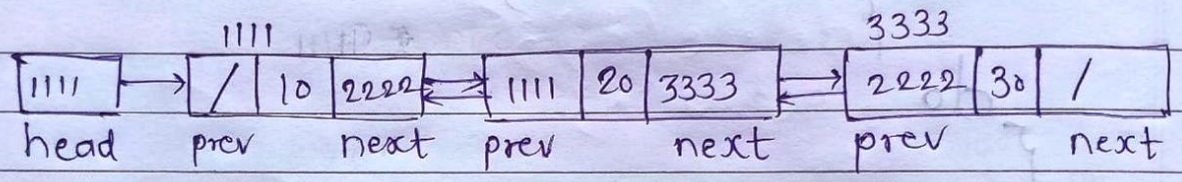
⇒ Search

```
int search (int x)
{
    Struct node *t=head;
        int found=0;
    if (head==0)
        found=0;
    else
    {
        do { if (t->info=x)
             {  found=1;
                break;
```

```
    }
        t = t -> link;
  } while (t != head);

    return found;
}
```

lecture 18

★ Doubly linked list



Structure :-
```
struct node
{
    struct *node *prev;
    int info;
    struct node * next;
};
```

⇒
```
void insert_at_begin (int x)
{
    struct node *p;
    p = new node;
    p -> prev = 0;
    p -> next = 0;
    p -> info = x;
    if ( head == 0)
        head = p;
    else
```

```
        p->next=head;
        head->prev=p;
        head=p;
    }
    return;
}
```

⇒ Insert At End

```
~~int~~ void insert_at_~~begin~~end(int x)
{
    struct node *p, ~~*q~~;
    p=new node;
    p->prev=0;
    p->next=0;
    p->info=x;

    if (head==0)
        head=p;
else
    {
        t=head;
        while(t->next!=0)
            t=t->next;
        t->next=p;
        p->prev=t;
    }
    return;
}
```

Date ___/___/_____

⇒ Insert At Position :-

```
void insert_at_pos (int pos, int x)
{
        struct node *p, *q, *t;
        int count=0;
        p=new node;
        p->prev=0;
        p->next=0;
        p->info=x;
    if (head==0)
    {
        cout<<" LL is empty"; delete p;
        return;
    }
    if (pos==1)
    {
        p->next=head;
        head->prev=p;
        head=p;
    }
else
{       t=head;
        count=1;
    while( count!=pos && t!=0)
{
        q=t;
        t=t->next;
        count++;
    }
```

Date ___/___/_____

Saathi

```
    if (t != 0)
    {
        q -> next = p;
        p -> prev = q;
        p -> next = t;
        t -> prev = p;
    }
else
        cout << "Invalid position";
    }
    return;
}
```

lecture 19
23-10-22

⇒ ① Delete At Begin()

```
int delete_at_begin
{
    Struct node *p;
    int x;
    if (head == 0)
    {
        cout << "LL is empty";
        return -1;
    }
    p = head;
    x = head -> info;
    head = head -> next;
    if (head != 0)
        head -> prev = 0;
```

Page No.

Date ___/___/___

```
        delete (p);
        return x;
    }
```

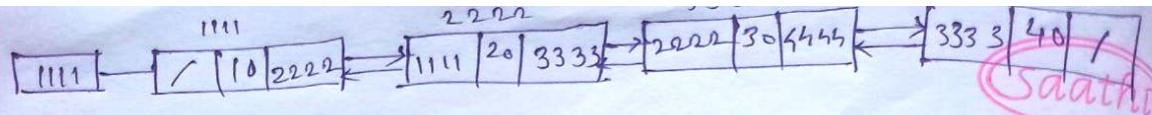
⇒ Delete At End

```
int delete_at_end()
{
    struct node *t, *q;
        int x;
    if (head==0)
    {
        cout<<"LL is empty";
        return -1;
    }
    if (head->next==0)
    {
        t=head;
        x=head->info;
        head=0;
    }
else
{
    t=head
    while (t->next!=0)
    {
        q=t;
        t=t->next;
    }
    q->next=0;
    x=t->~~next~~ info;
```

```
}
        delete (t);
        return x;
}
```

⇒ Delete At Position

```
int delete_at_Position ()
{
        struct *q, *t;
        int x, count=0;
        if (head==0)
        {
                cout<<" LL is empty";
                return -1;
        }
    if (pos==1)
{
        t=head;
        x=head->info;
        head=head->next;
        if (head !=0)
                head->prev=0;
        delete(t);
        return x;
    }
        t=head;
        count=1;
        while (count!=pos && t!=0)
        {
                q=t;
```



Date ___/___/___

```
        t = t -> next;
        count ++;
    }
    if (t != 0)
    {
        x = t -> info;
        q -> next = t -> next;
    if ( q -> next != 0)
        q -> next -> prev = q;

        delete (t);
        return x;
    }
else
    { cout << " Invalid position";
        return -1; }
}
```

⇒ Display

⇒ Count ~~void~~

⇒ search.

lecture 20

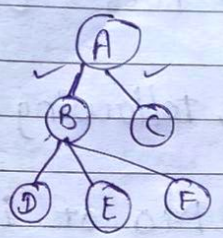
23-10-22

Unit 4
Trees.

If n is no of nodes, then edges = n-1

① Degree of node.

No of subtrees of node is called degree of a node



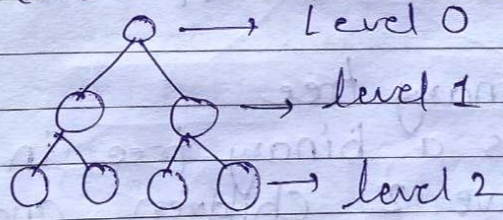
A has B, C subtrees
B has 3 subtrees D, E, F

② Degree of tree.

The node having maximum degree in tree is ~~called~~ the Degree of tree

e.g Degree of tree is 3

③ Level of tree.



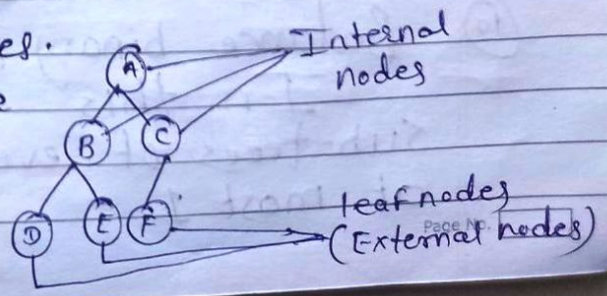
④ Height of tree

Maximum level is height of tree

e.g Height of above tree is 2.

⑤ Internal & External nodes.

Nodes having child node is called internal nodes.



Date ___/___/___

(6) Binary tree

A binary is a non linear data structure in which each node has at most two children which reffered as left and right child.

(7) Full (Strict) Binary tree

Every node has 0 or 2 children.

For full binary tree, following eqn is alway true

No of Leaf nodes = No of internal nodes + 1

Note: Full binary always have odd no. of nodes

(8) Complete Binary tree

All levels completely filled with nodes except the last level and in the last level, all the nodes are as left side as possible.

(9) Perfect Binary tree

It is a binary tree in which all internal nodes have 2 children and the leaf nodes are at the same depth or same level

Total no of nodes in perfect binary tree with height H is $2^{(H+1)} - 1$

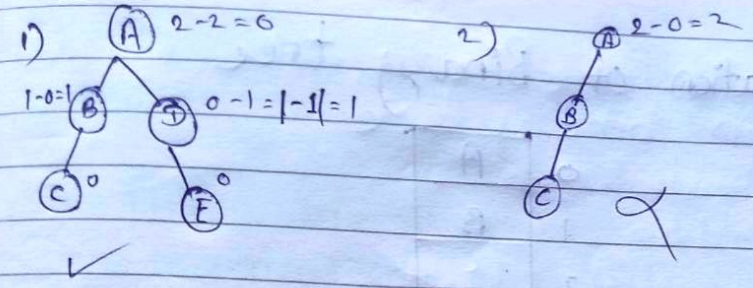
(10) Balance binary tree

In this height of the left and right sub-trees of every node may differ by at most 1

Page No.

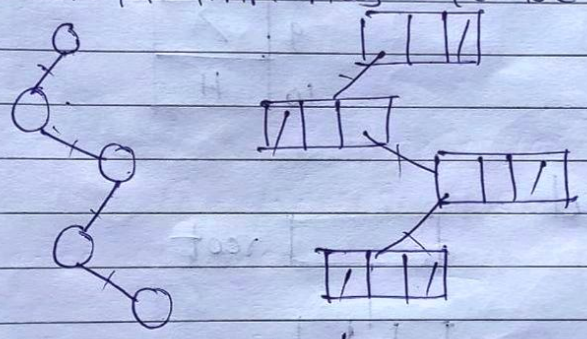
Date ___/___/___

Saathi



(11) Degenerate binary tree.

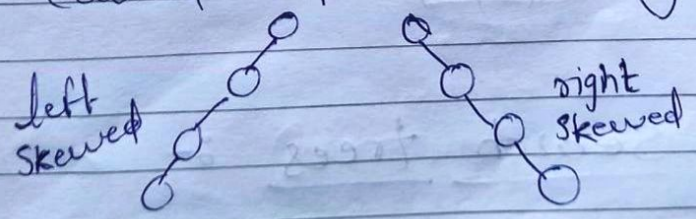
A binary tree where every parent node has only one child node. ~~three~~ such trees are like link list. Even slower as both left and right child link has to be checked



Height Of Degenerate tree is equal to ~~total number of nodes in that tree~~ = Total nodes - 1

(12) Skewed Binary tree.

A tree that is dominated solely by either left child nodes or right child nodes is called skewed binary tree

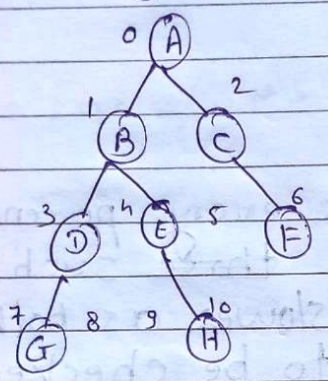


Page No.

Date ___/___/___

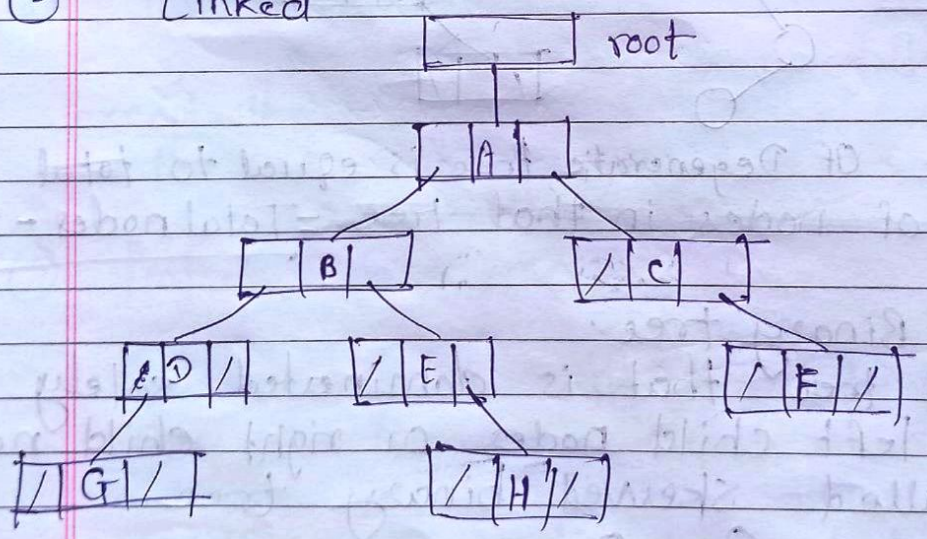
⇒ Representation of Binary tree

① Array.



0	A
1	B
2	C
3	D
4	E
5	/
6	F
7	G
8	/
9	/
10	H

② Linked



⇒ Binary Search trees :-

* ~~Operatio~~ It is a tree in which all the nodes follow the below-mentioned properties

Page No.

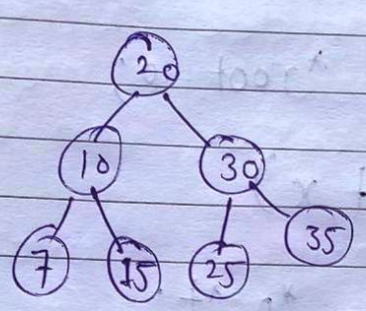
Date ___/___/_____

Saathi

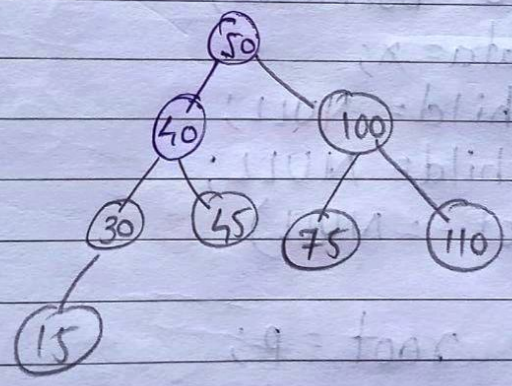
① The value of left subtrees is less than the value of its parent

② The value of right subtrees is greate than or equal to the value of it parents value

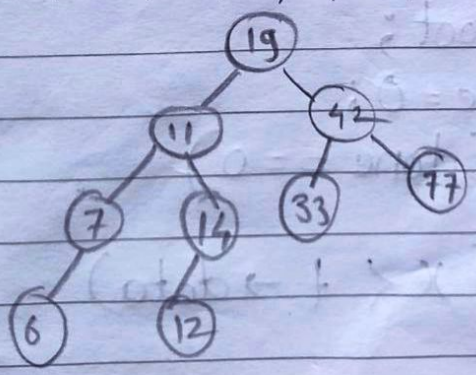
e.g Construct bst from data
20, 10, 7, 30, 15, 35, 25



50, 40, 30, 45, 100, 15, 75, 110



19, 42, 77, 33, 11, 7, 6, 14, 12



Date ___/___/_____

★ Binary search tree program

⇒ Insert in binary tree

```
Struct bstnode
{
    struct bstnode *lchild;
    int data;
    struct bstnode *rchild;
};

    Struct bstnode *root=0;

void insert(int x)
{
    Struct bstnode *p, *t;
    int done;
    p=new bstnode;
     p->data=x;
     p->lchild= NULL;
     p->rchild= NULL;
    if (root==NULL)
    {
            root=p;
    }
else
  {     t=root;
        done=0;
    while (done==0)
    {
        if (x< t->data)
        {
```

```
        if (t->lchild == NULL)
        {
            t->lchild = p;
            done = 1;
        }
        else
            t = t->lchild;
    }
else
    {
        if (t->rchild == NULL)
        {
            t->rchild = p;
            done = 1;
        }
        else
            t = t->rchild;
    }
    } end while
} end if else
}
```

BST

⇒ Delete from binary tree.

When we delete node from binary tree there are three posibilities

(i) Node to be deleted is leaf

→ Simply remove the node

Date ___/___/___

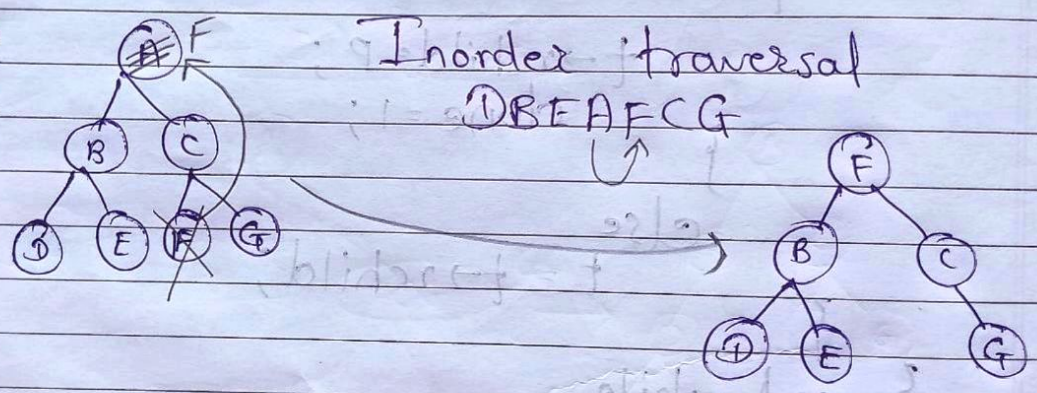
② Node to be deleted has only one child

→ Copy the child to the node & delete the child

③ Node to be deleted has two children

Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor (inorder predecessor can also be use)

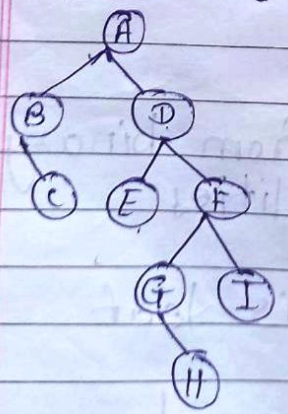
delete A, having 2 child



Inorder traversal
DBEAFCG

⇒ Traversal technique

① Inorder traversal (LDR)




```
void inorder(struct bstnode *t)
{
    if (t!=NULL)
    {
        inorder(t->lchild);
        cout<< t->data;
        inorder (t->rchild);
    }
}
```

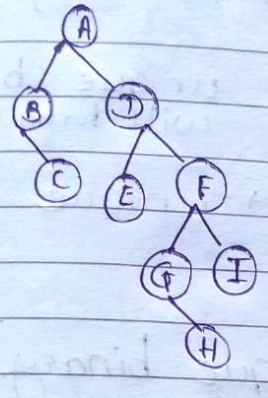
root

Inorder traversal- BCAE D ~~HG~~ F I GH

Date ___/___/_____

Saathi

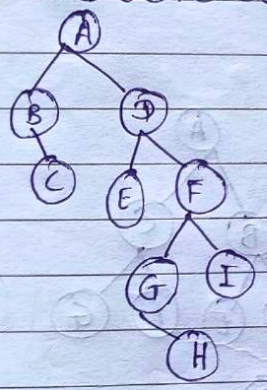
② Preorder traversal (DLR)




```
void preorder(struct bstnode *t)
{
    if (t!=NULL)
    {
        cout << t->data;
        preorder(t->lchild);
        preorder(t->rchild);
    }
}
```

Preorder
ABCDEFGHI

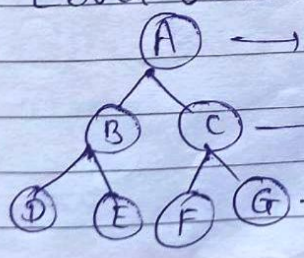
③ Postorder traversal (LRD)




```
void postorder(struct bstnode *t)
{
    if (t!=NULL)
    {
        postorder(t->lchild);
        postorder(t->rchild);
        cout << t->data;
    }
}
```

Postorder
CBEHGIFDA

④ Level order traversal



ABCDEFG

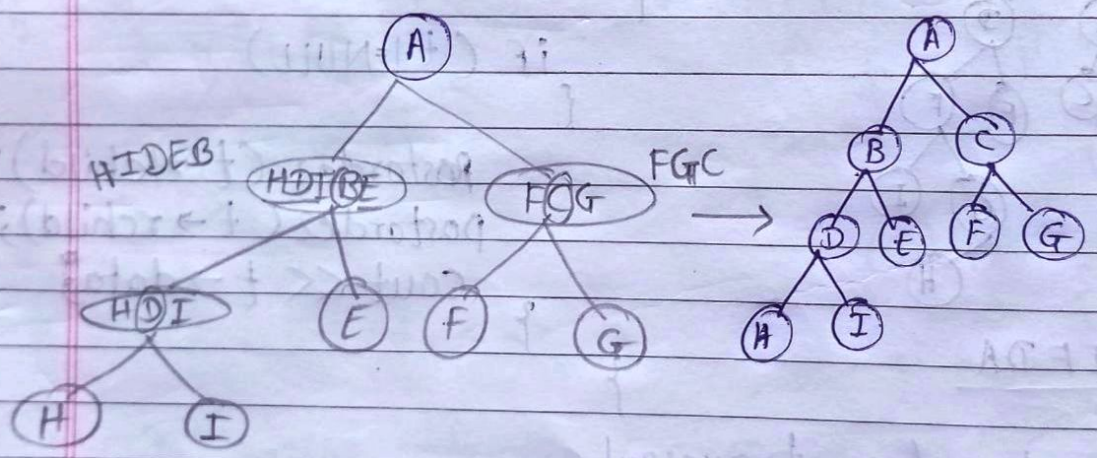
Page No.

Date ___ / ___ / _____

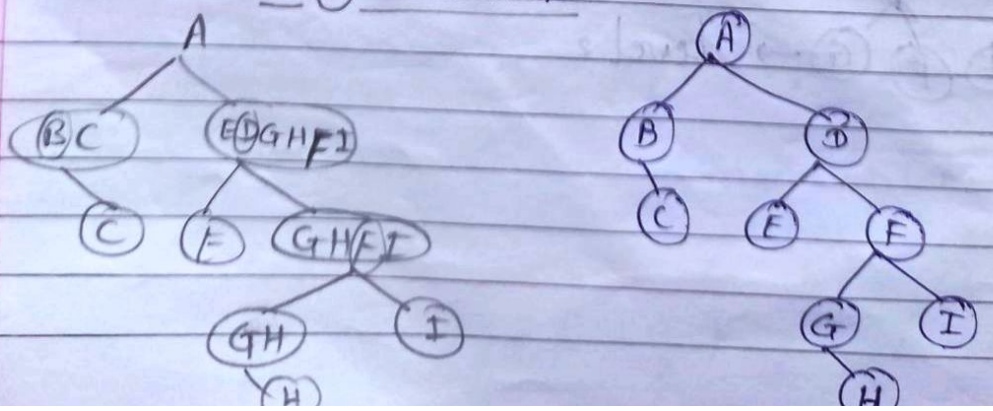
⇒ Construction of Binary tree from given traversals

1. Inorder and Preorder → unique binary tree will form
2. Inorder and Postorder → unique
3. Preorder and postorder
   (Does not guarantee a unique binary tree if the tree is not a full binary tree)

e.g 1 Postorder : HIDEBFGCA
Inorder : HDIBEAFCG



e.g 2 Preorder : ABCDEFGHI
Inorder : BCAEDGHFI



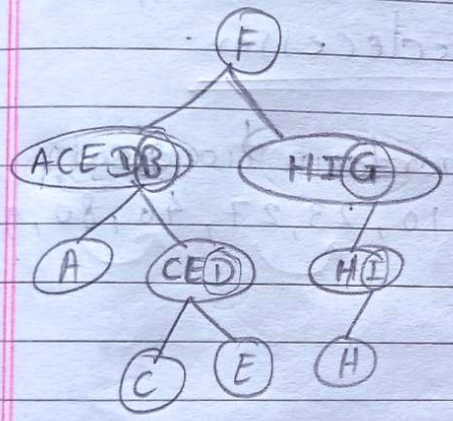
Page No.

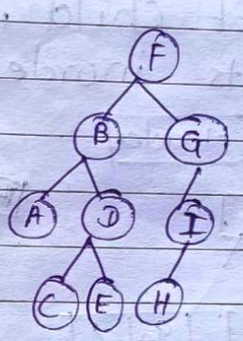
Date ___/___/_____

Saathi

③ e.g Preorder : F B A D C E G I H
Postorder : A C E D B H I G F

① Find root
② Find successor of root inpreorder
③ To the left of successor in postorder including itself are left subtree & to the right are right subtree.





★ Threaded binary trees.

① Binary tree may have at most two children

② But if they have only one child, or no children, the link part in the linked list representation remain null

③ Using threaded binary tree representation, we can reuse that empty links by making some threads.

There are two types of threaded binary tree

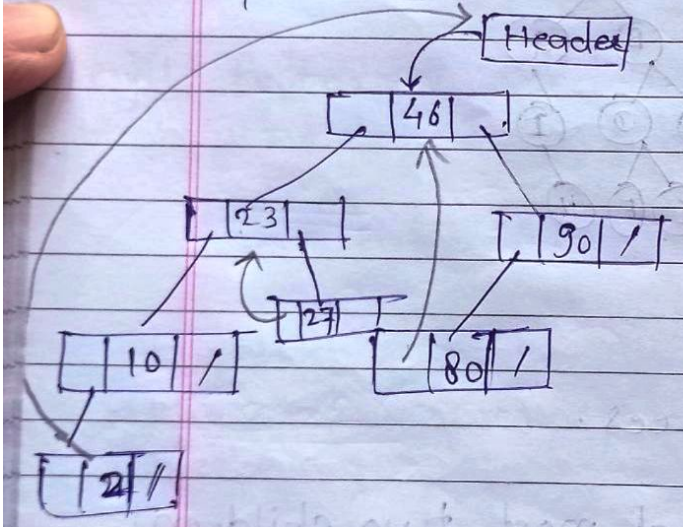
Page No.

Date ___/___/___

① Single threaded tree (Two variations Left threaded or Right threaded)

② Fully threaded binary tree

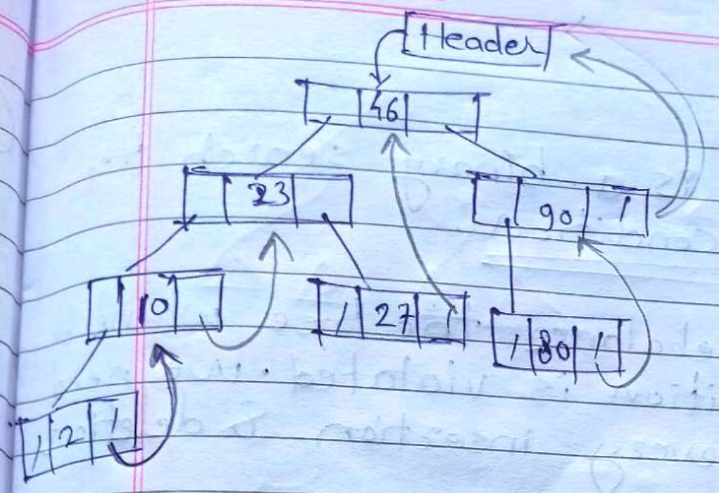
1) Single threaded tree (Left threaded tree)
In the left threaded mode if some node has no left child, then the left pointer will point to its inorder predecessor.



Header inorder traversal
2, 10, 23, 27, 46, 80, 90

② Single Threaded Tree (Right threaded tree)
In the right threaded mode if some node has no right child, then the right pointer will point to its inorder successor.

Inorder Traversal : Header
2, 10, 23, 27, 46, 80, 90



2] Fully threaded Tree:

In order to ~~add~~ differentiate betn whether point is link or thread two additional fields are added to threaded trees.

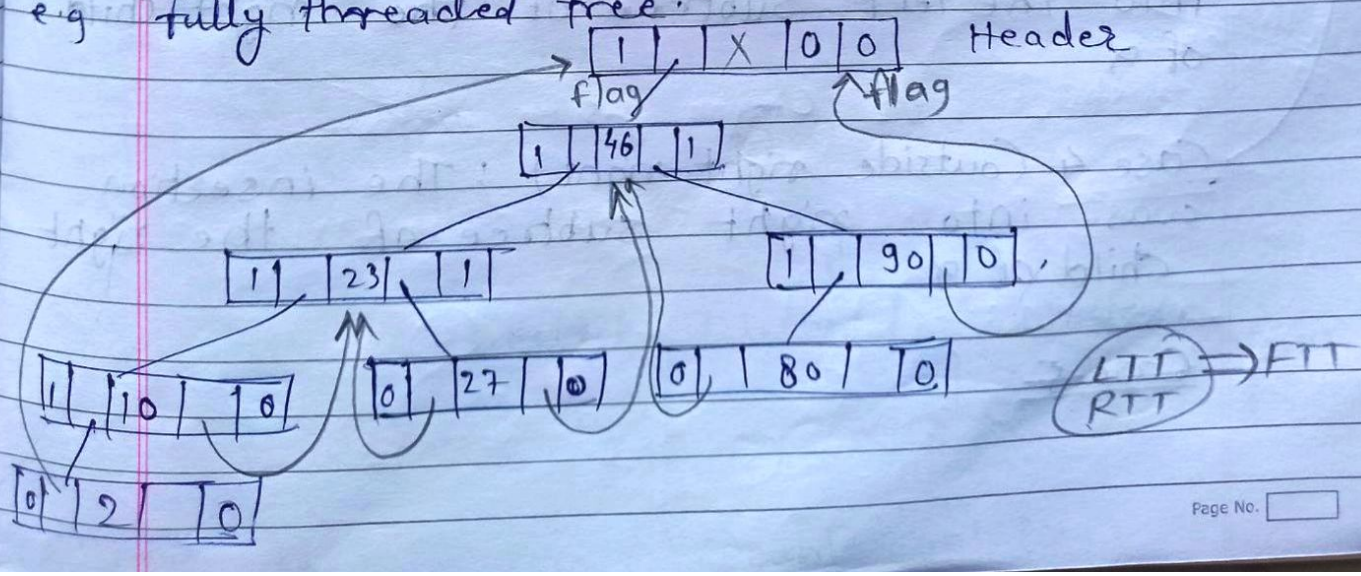
left flag	left link	Data	right link	right flag

1	56	0

1 indicate link
0 indicates a thread

Note: Combination of Right & left threaded is called Fully threaded tree

e.g fully threaded tree.



lecture 22
26-10-2022

Saathi

⇒ AVL tree (

An AVL tree is a binary search tree with a balance condition.

We need to rebalance the tree whenever the balance condition is violated. We need to check after every insertion & deletion.

Node where violation occurs is called α node

★ Balancing condition is violation cases:

Case 1 (outside left-left) : The insertion was into left subtree of the left child of α.

Case 2 (inside left-right) : The insertion was into the right subtree of the left child of α.

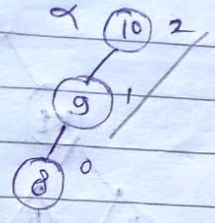
Case 3 (inside right-left) : The insertion was into the left subtree of the right child of α

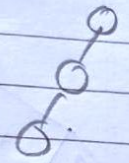
Case 4 (outside right-right) : The insertion was into right subtree of the right child of α

~~Case 5~~

Date ___/___/___

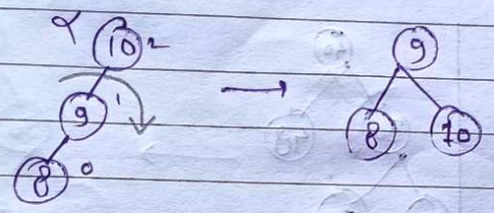
Case 1 : α & its nearest neighbour has more importance.



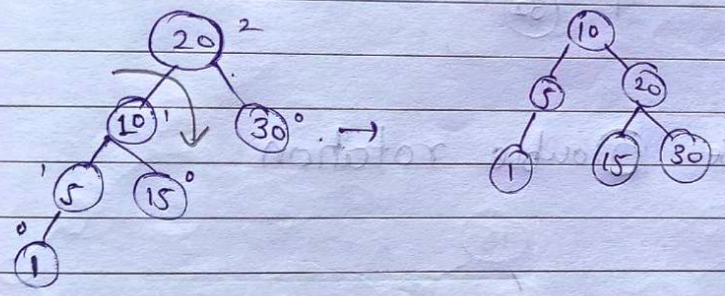


Solution is use <u>single rotation</u>

Balancing Bottom up करायची



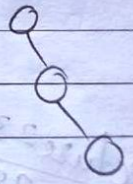
e.g

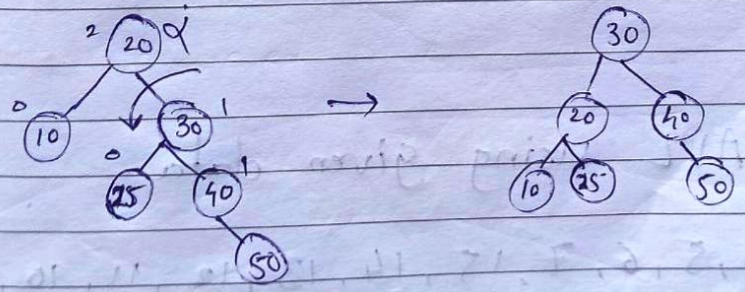


← Now Balance, so it is AVL tree.

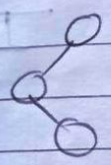
Case 4 :

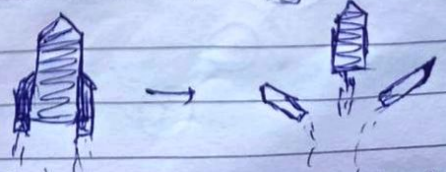
Solution is single rotation.





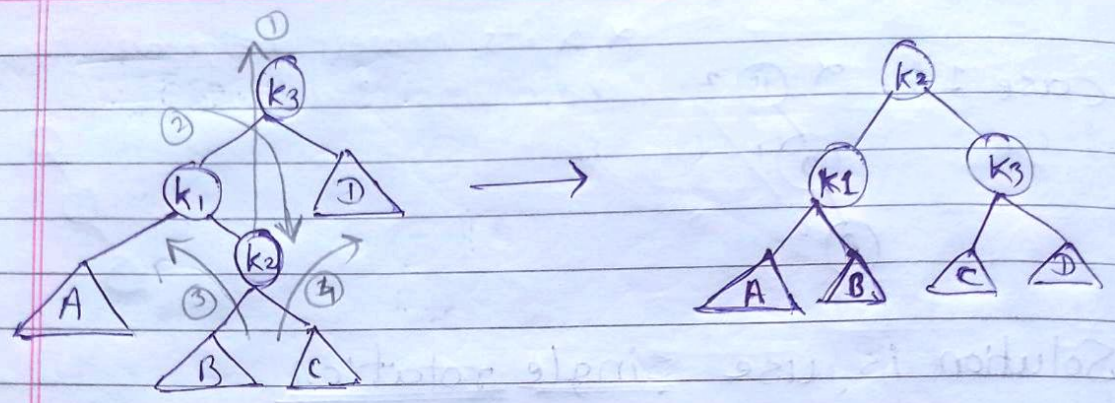
Case 2 : [Rocket Method] ← Trick



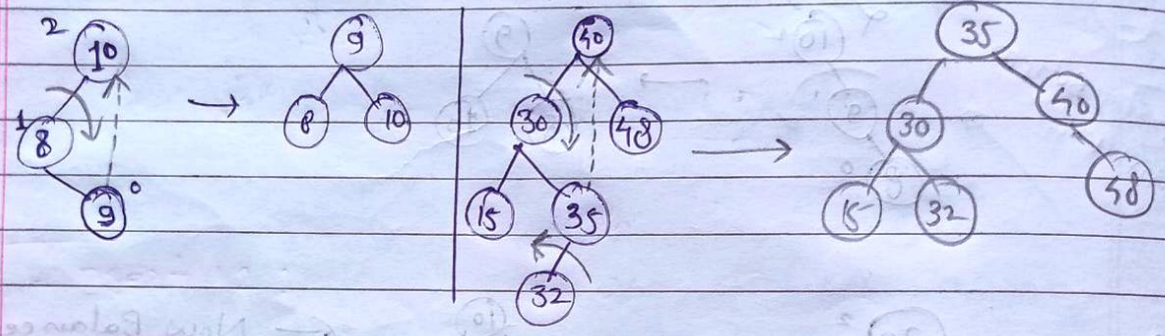


To solve use Double rotation

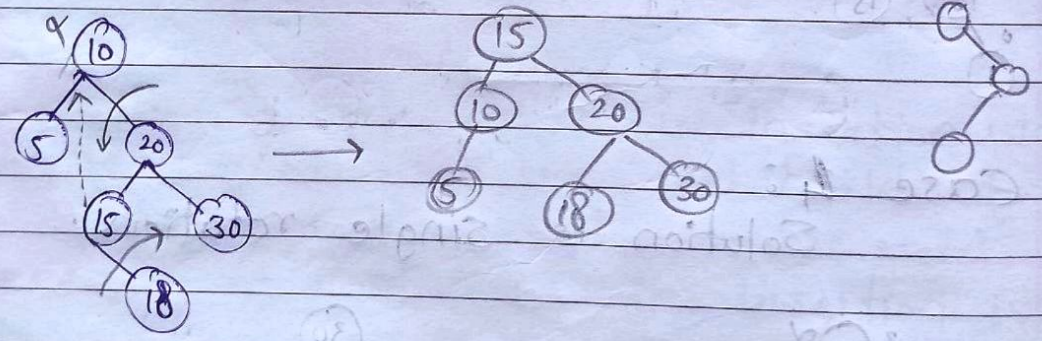
Page No.



e.g



Case 3: Use Double rotation

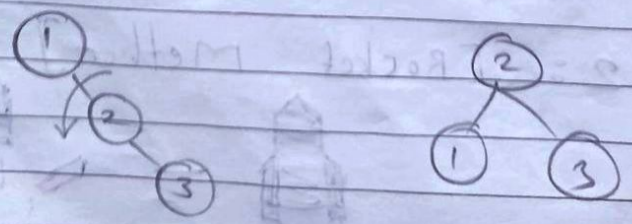


lecture 23
27-10-22

Construct AVL using given data.

1, 2, 3, 4, 5, 6, 7, 15, 14, 13, 12, 11, 10, 9, 8.

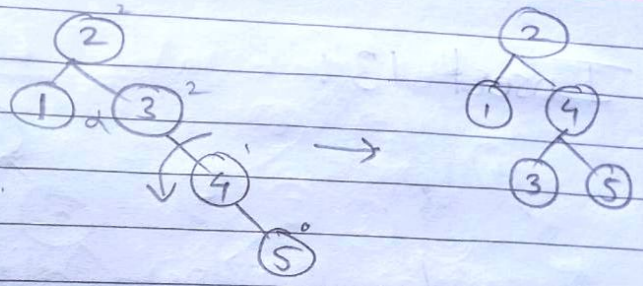
Insert 1, 2, 3:



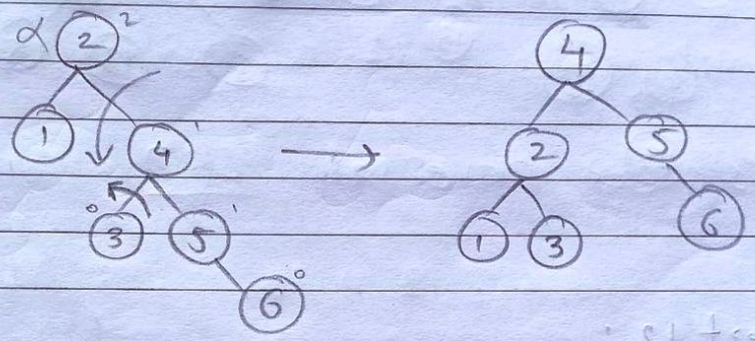
Date ___/___/_____

Saathi

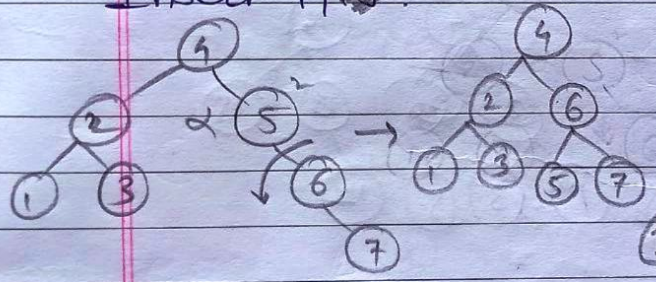
Insert 4, 5 :



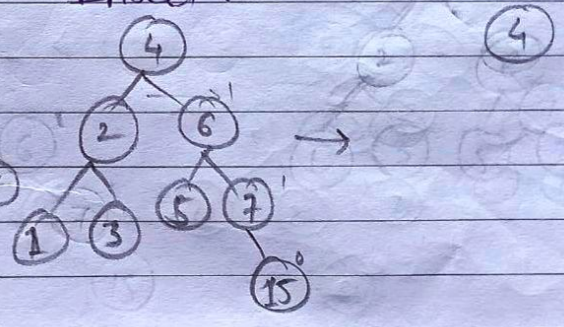
Insert 6 :



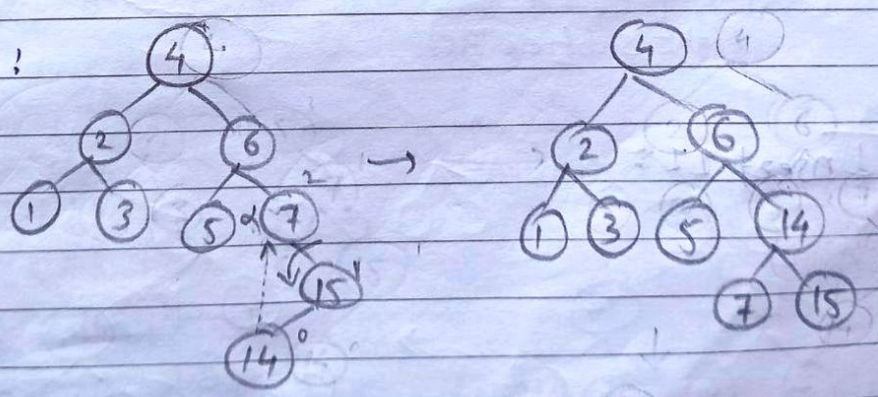
Insert 7, 15 :



Insert 15 :

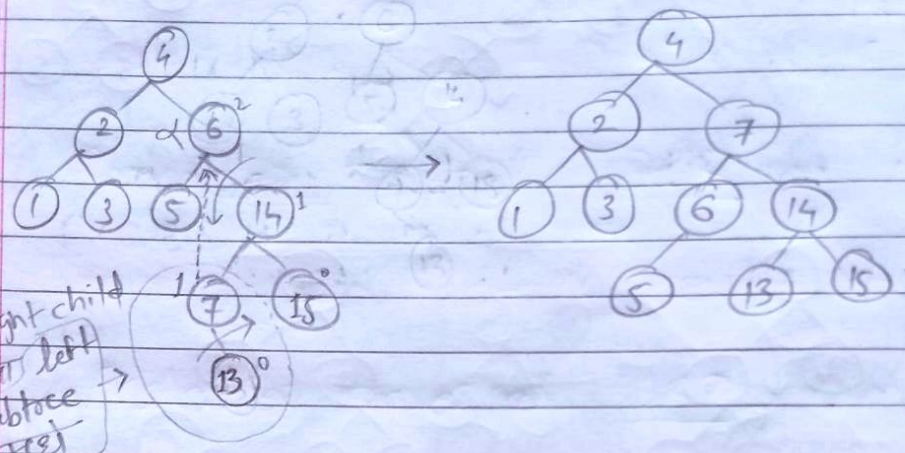


Insert 14 :

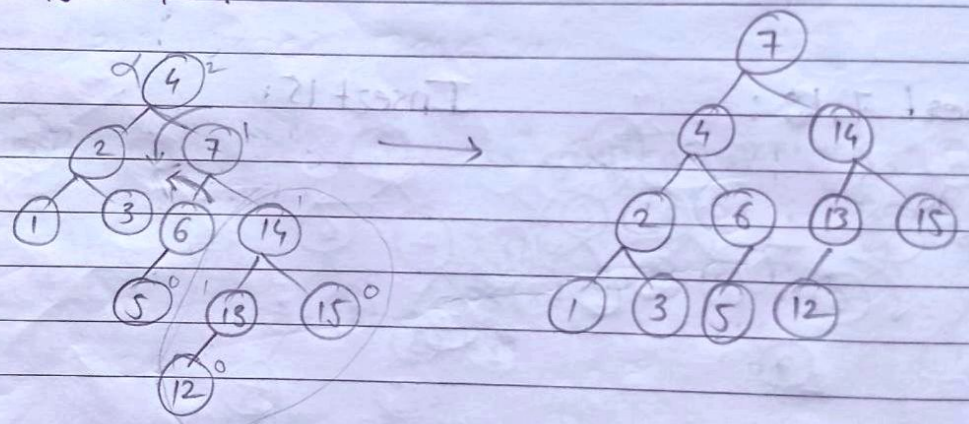


Date ___/___/_____

Insert 13 :

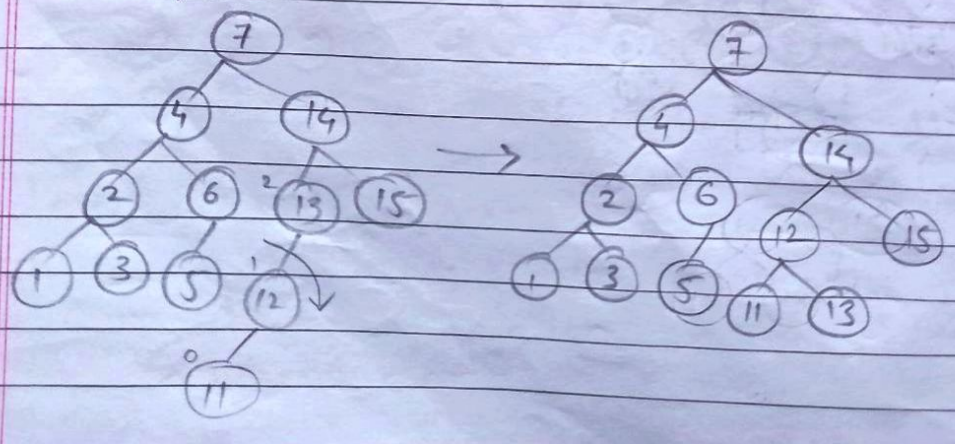


Insert 12 :



Right child (7), च्या
Right subtree मध्ये

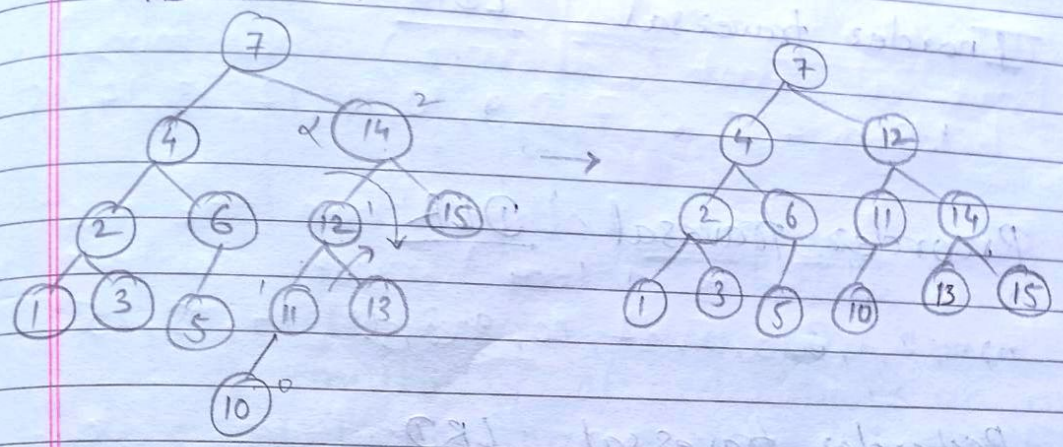
Insert 11 :



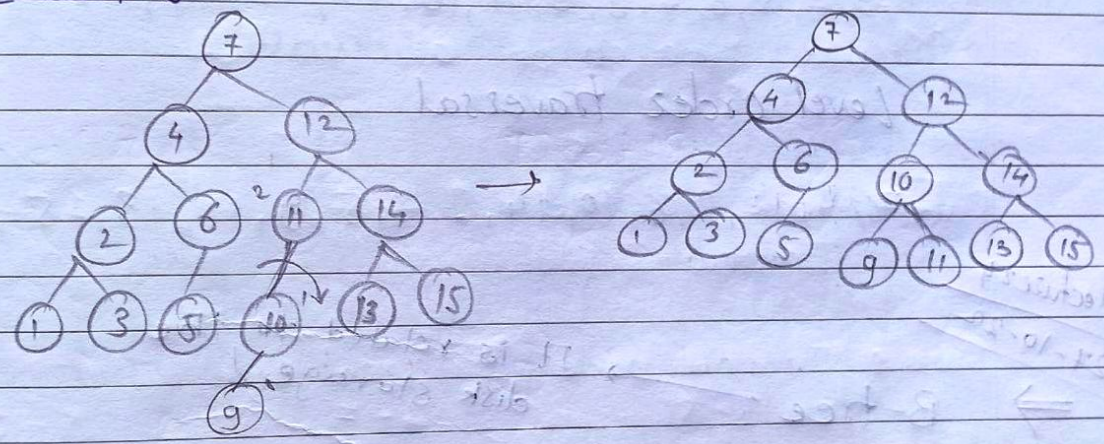
Date ___/___/_____

Saathi

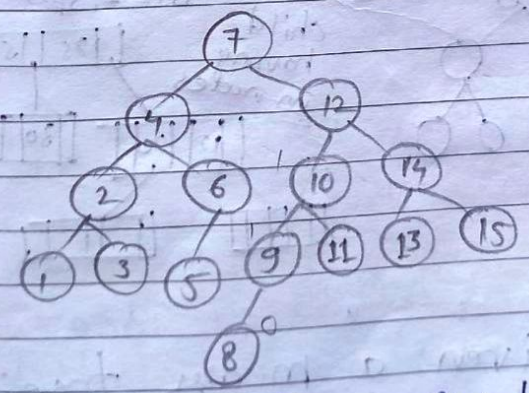
Insert 10 :



Insert 9 :



Insert 8 :



final

AVL tree

Note : Every AVL tree is binary search tree.

BST को छोटा करने का formula है AVL tree

Date ___/___/_____

Inorder traversal **LDR**

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

Preorder traversal **DLR**

7, 4, 2, 1, 3, 6, 5, 12, 10, 9, 8, 11, 14, 13, 15

Postorder traversal **LRD**

1, 3, 2, 5, 6, 4, 8, 9, 11, 10, 13, 15, 14, 12, 7

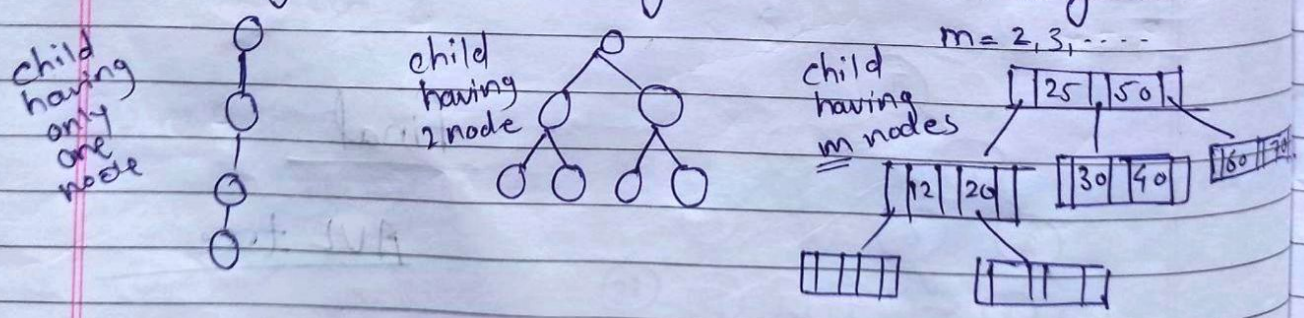
Level order traversal

7, 4, 12, 2, 6, 10, 14, 1, 3, 5, 9, 11, 13, 15, 8

Lecture 24
27-10-22

⇒ B-tree → It is related to the disk storage

Unary tree Binary tree m-ary tree



If you are given a m-ary tree: find number of children & keys for the following depth.

m	Max children per node	keys
5	5	4
8	8	7
15	15	14

1) Defination:

A multiway of order m is an ordered tree where each node has at most m children.

e.g Multiway search tree of order 4

★ Rules for B-tree :-

1) The root node can have as few as (minimum) 2 children if it is an internal node, and can obviously have no children if the root node is a leaf (that is, the whole tree consists only of the root node).

ceil(3.5)↑ = 4

if m = 5 then

ceil(m/5) = 3

2) All internal nodes (except the root node) have at least ceil(m/2) (nonempty) children.

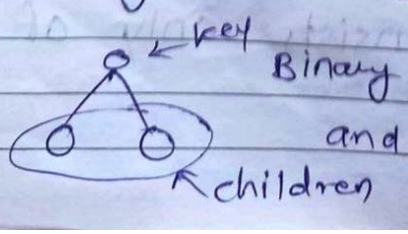
3) Each leaf node (other than the root node if it is a leaf) must contain at least ceil(m/2) − 1 keys.

4) All leaves are on the bottom level.

M-way search of of order.

	5	7	10
Rule 1	Leaf – 0 child Internal – 2 child	〃	〃
Rule 2	Internal node 3 – 5	Internal node 4 – 7	〃 5 – 10
Rule 3	~~At least key~~ keys 2 – 4	keys 3 – 6	keys 4 – 9
Rule 4	×	×	×

Use Binary tree as a e.g



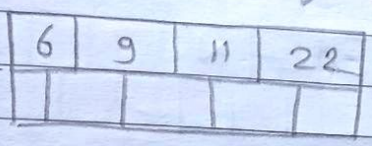
Binary tree has 2 children and 1 key

Date ___/___/___

Saathi

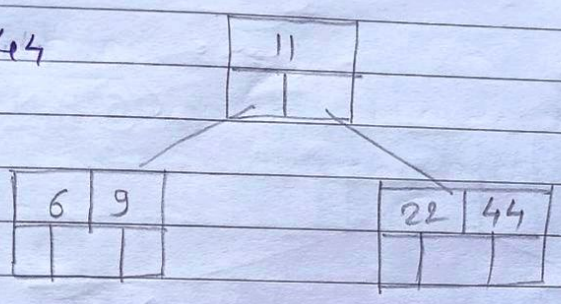
Construct a B-tree of order 5 for the data 11, 9, 6, 22, 44, 92

Insert 11, 9, 6, 22



6 9 (11) 22 44

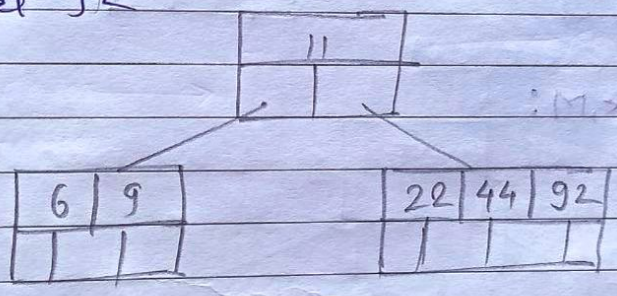
Insert 44



← leaf keys

ceil $[5/2] - 1 = 2$

Insert 92



lecture 25

Construct B tree of order 5 using data:
C N G A H E K M Q F W L T Z D P R X Y S

Internal nodes : ceil $[5/2] = \lceil 2.5 \rceil = \underline{\underline{3}}$ children (at least)

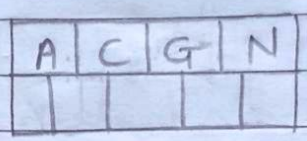
Leaf nodes : ceil $[5/2] - 1 = \lceil 2.5 \rceil - 1 = 3 - 1 = \underline{\underline{2}}$ keys (at least.)

Page No.

Make sure all ~~nod~~ leaf nodes are at Bottom level throughout the whole process

Date ___/___/___

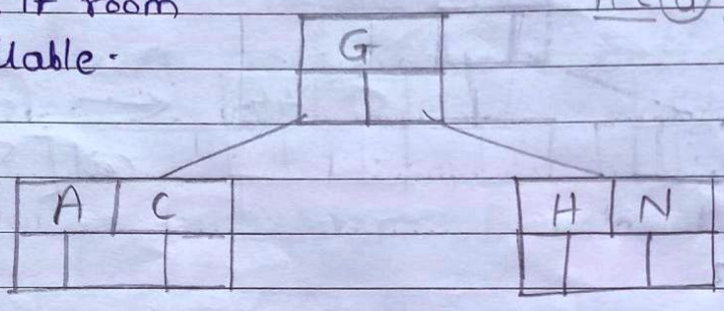
Insert C, N, G, A : Because Maximum 4 keys can be insert in B-tree



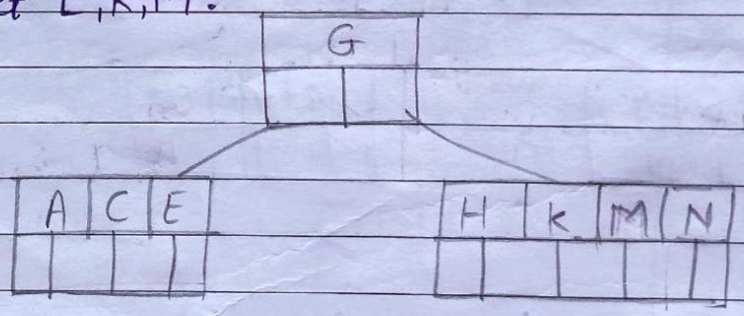
Insert H :

Split nodes if room is not available.

A C G H N



Insert E, K, M:



Insert Q :

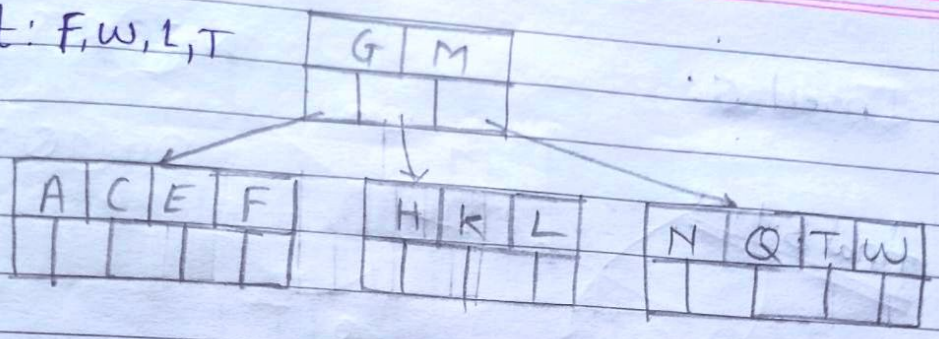
H K M N Q

split

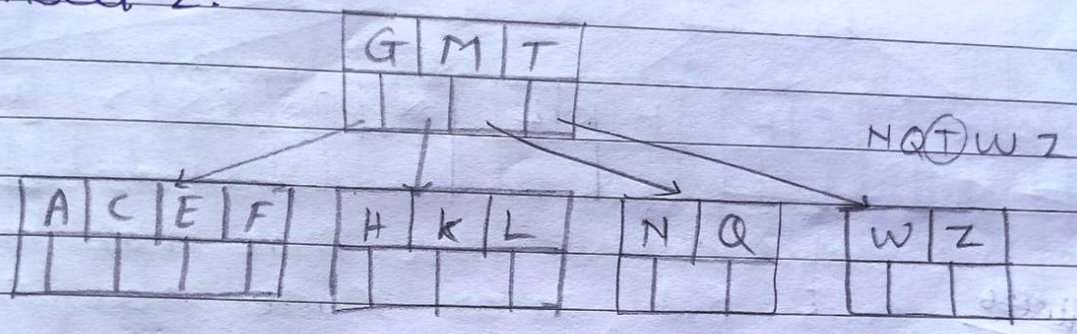
Date ___/___/___

Saathi

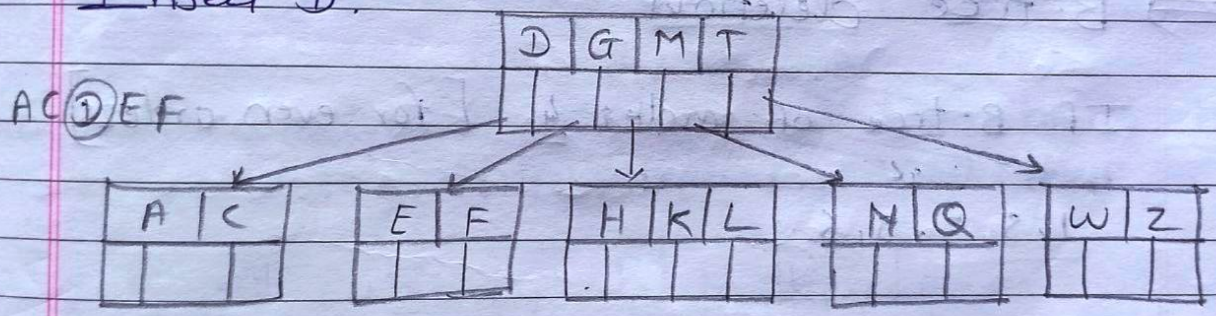
Insert: F, W, L, T



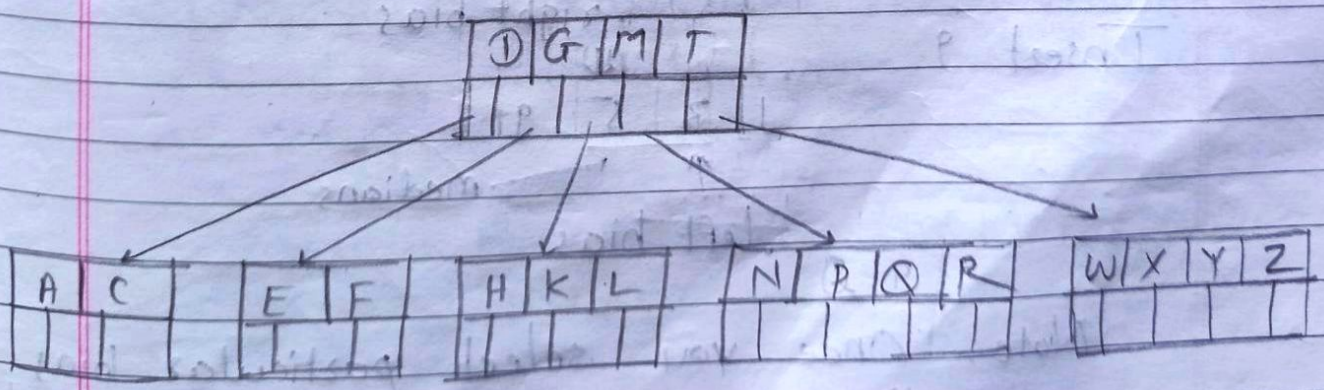
Insert Z:



Insert D:



Insert P, R, X, Y:



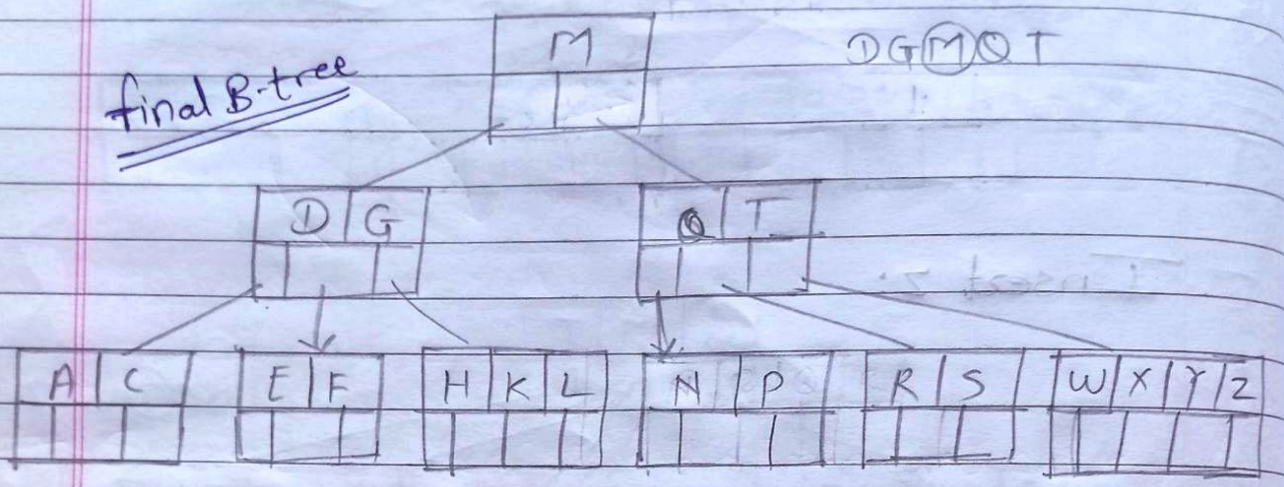
Page No.

Insert S:

NPQRTS

DGMQT

final B-tree

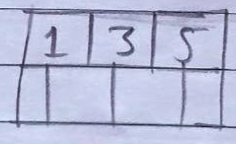


~~lecture 6~~

⟹ ~~B-tree~~ ~~deletion~~

If B-tree of order 4 [for even orders]

1, 5, 3, 9, 6



Insert 9

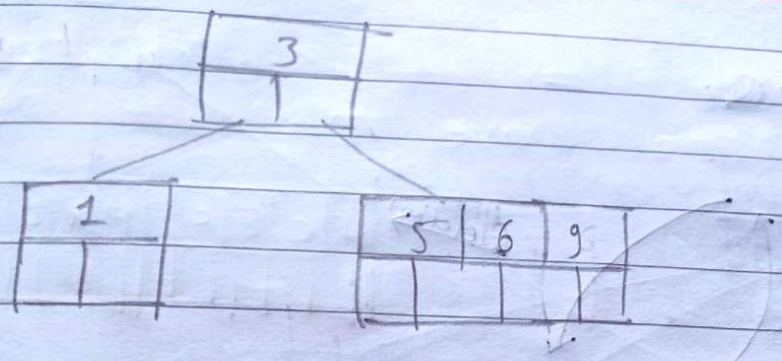
right bias

1 3 5 9

left bias

medians

Note: Once you select particular bias then you must select only left bias / right bias throughout whole B-tree



Lecture 26

⇒ B-tree deletion.

There are two cases when we try to delete node from B-tree:

★ Case 1: key to be deleted is in leaf node.

Subcase A: Leaf node contains more than the minimum no of keys:

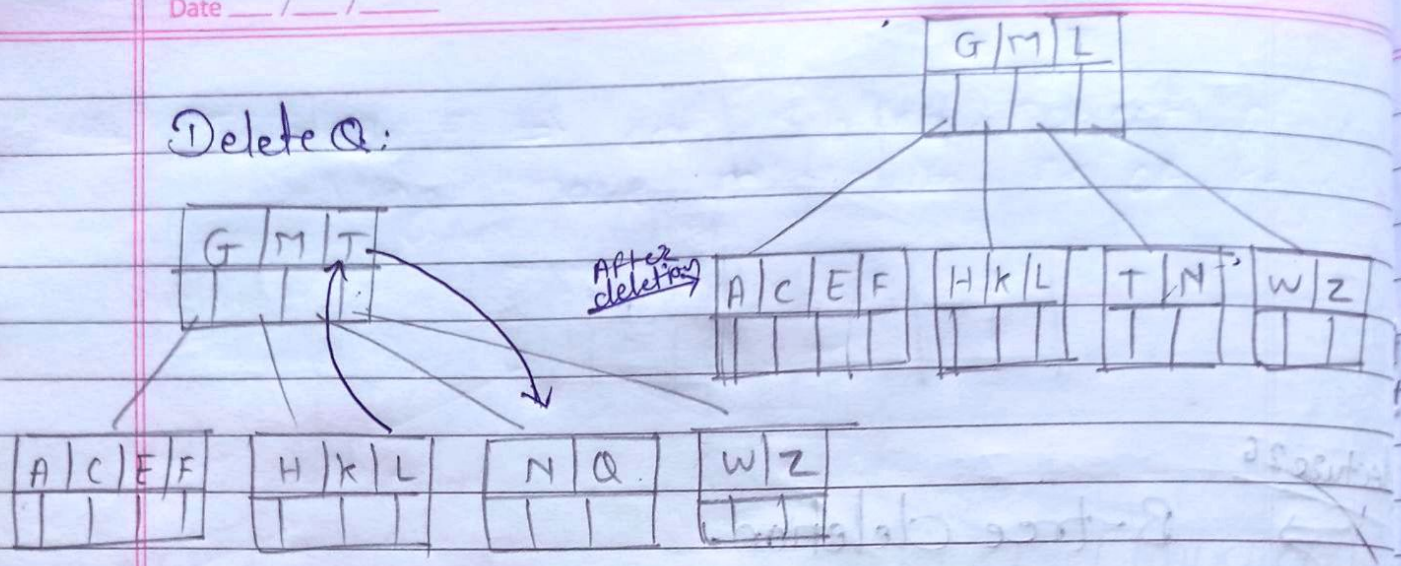
Sol^n: Simply delete key.

Subcase B: leaf node has exactly minimum no of keys. & sibling has more than min. no keys

Sol^n: Borrow a key from left/Right sibling.

Date ___/___/___

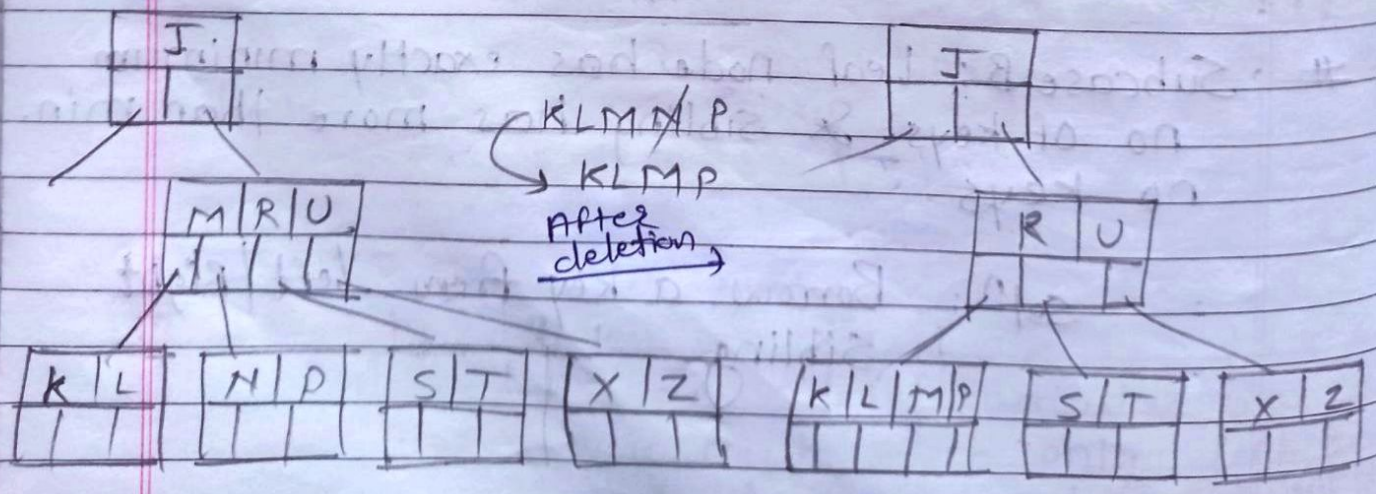
Delete Q:



Subcase C: Leaf contain min. no of key. & Siblings do not have more than minimum keys. But parent node has more than minimum no of keys.

Sol^n: Merge left or right sibling along with the corresponding key from the parent with the leaf node. Then delete the desired key.

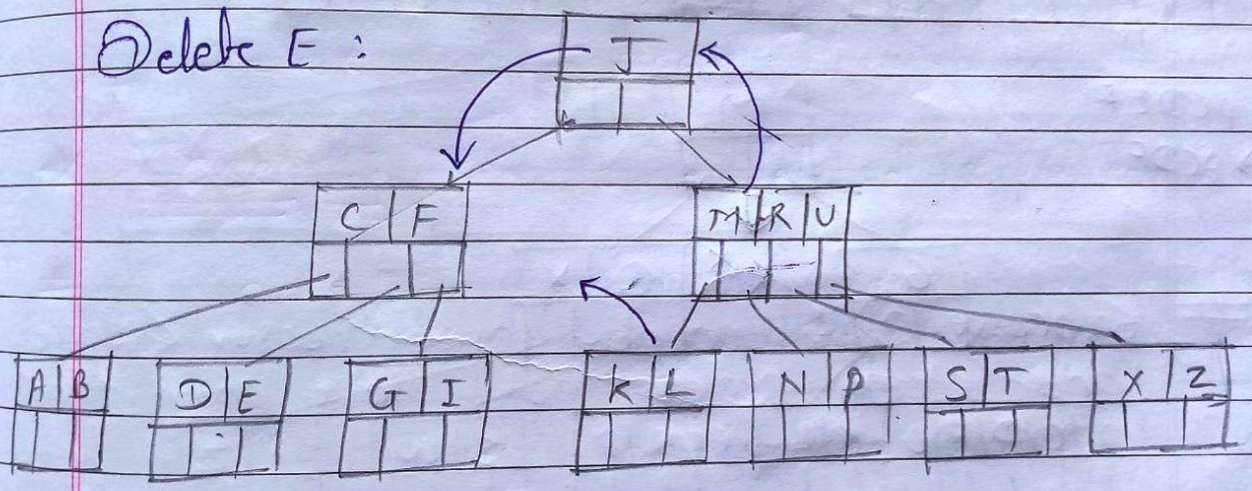
Delete N:



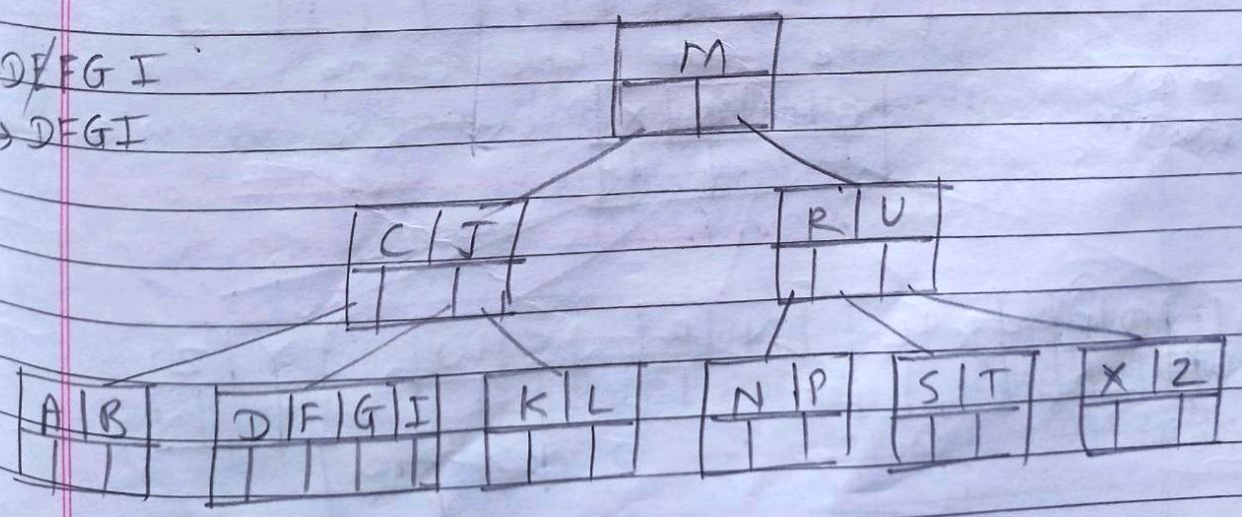
Subcase D: Leaf node, left/right siblings, & parent node do not have more than minimum no of keys. But sibling of parent has more than min. no of key.

Sol^n: Borrow key from parent ~~sibling~~ ~~The~~ & merge left or right sibling. Then delete the key., Now parent remains with one key short, it will borrow this key from its siblings through its parent

Delete E :

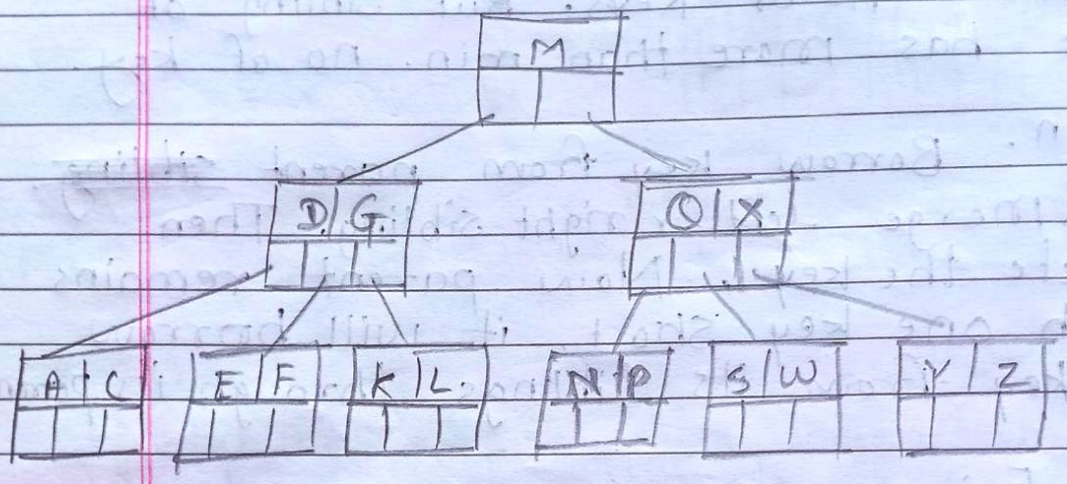


~~DEGI~~
DEGI

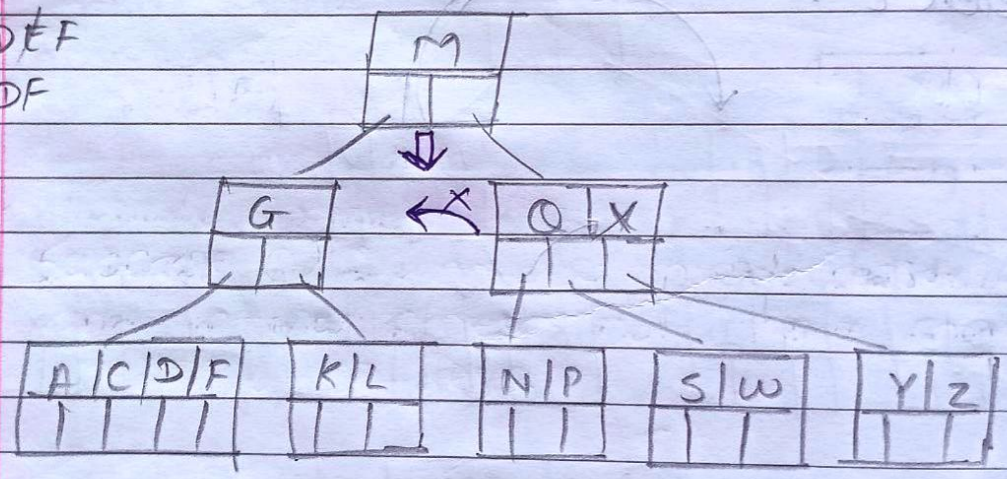


Date ___/___/_____

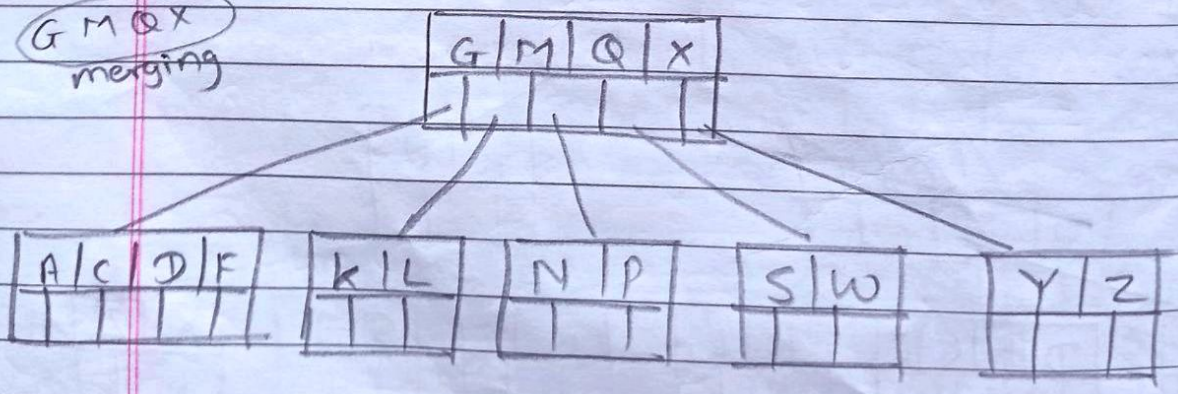
Subcase E ⇒ leaf, sibling, parent, parent sibling have minimum no of keys.



ACDEF
→ ACDF



GMQX
merging



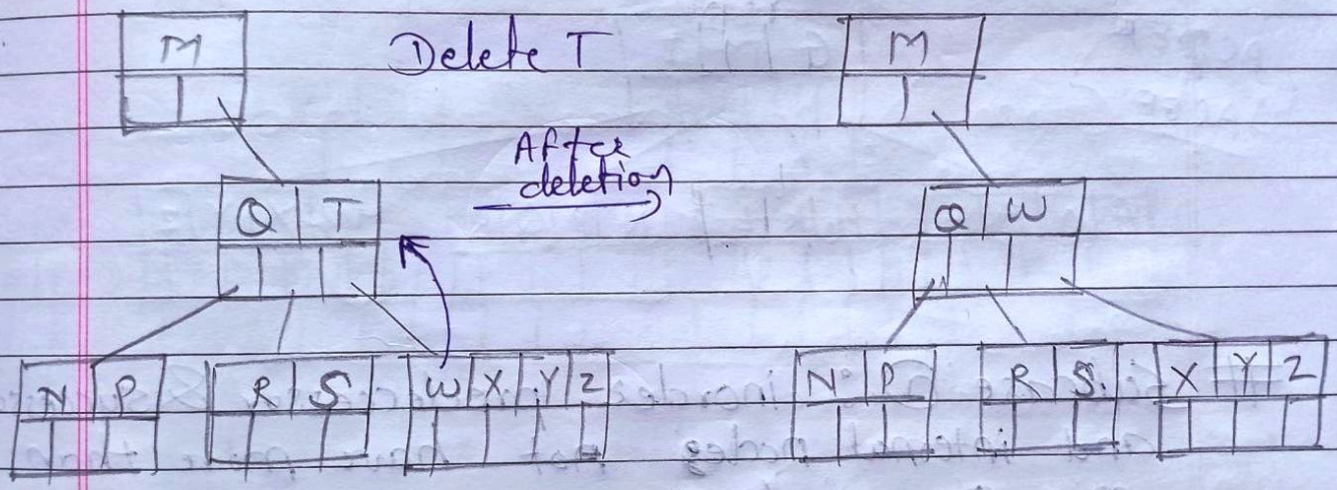
Date ___/___/___

Saathi

Case 2: Key to be deleted is in internal node.

Subcase A : If the node containing inorder successor has more than the minimum no of keys:

Sol^n: Borrow the inorder successor from right subtree.



Subcase B: Inorder predeccessor node has more than the minimum number of keys:

Sol^n: Borrow key from inorder successor

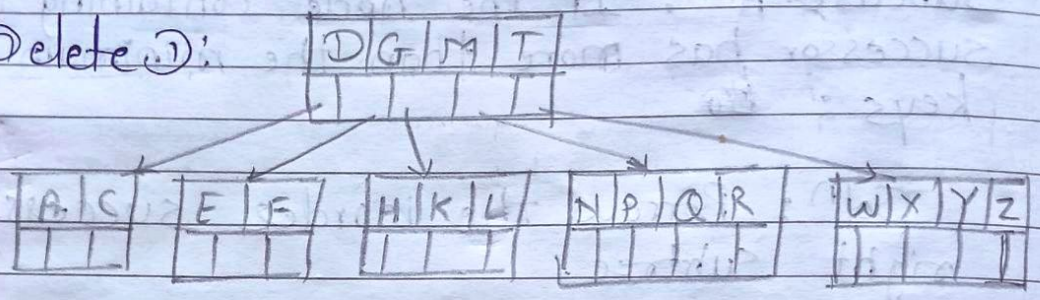
Subcase C: Both nodes containing inorder predeccessor & successor has minimum no of keys.

Date ___/___/___

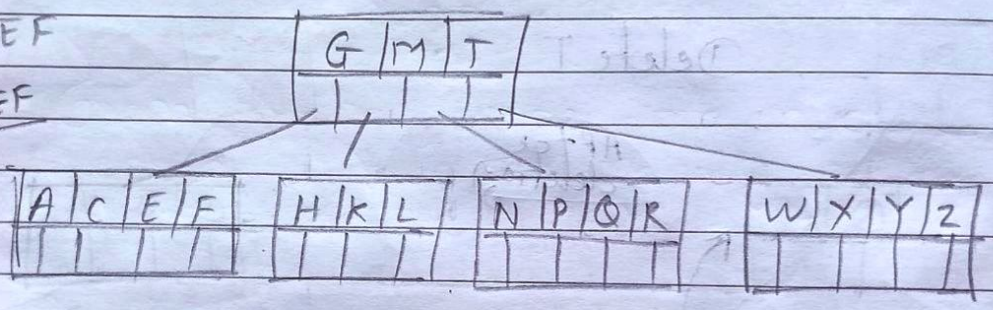
Saathi

Sol^n: Merge both nodes (i.e inorder predeccessor & successor) along with the desired key that is going to be deleted and then delete the key.

Delete D:



ACDEF
ACEF

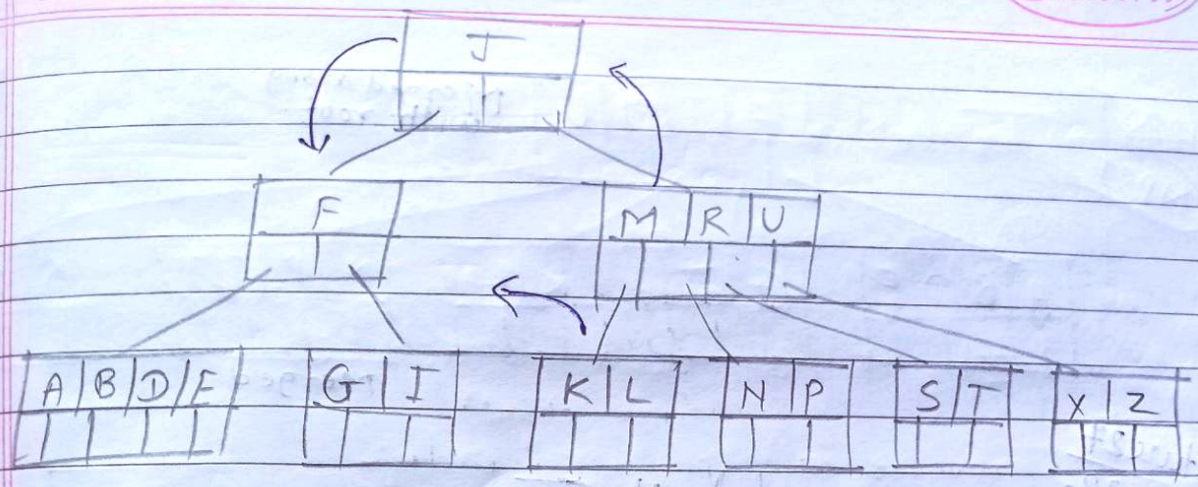


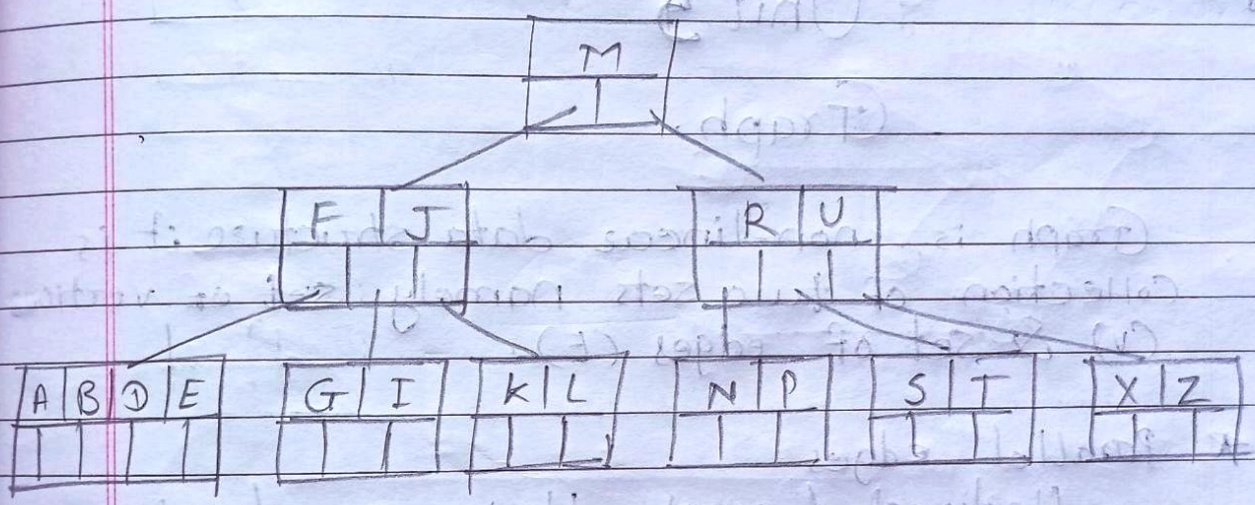
Subcase ①: inorder predeccessor & successor and internal nodes not have more than min no of keys.

Sol^n: Merge ~~leaf~~ node along with key. Then delete key. Now the internal node is short of key, it will ~~seek t~~ borrow this key from its sibling through parent.

Delete C:

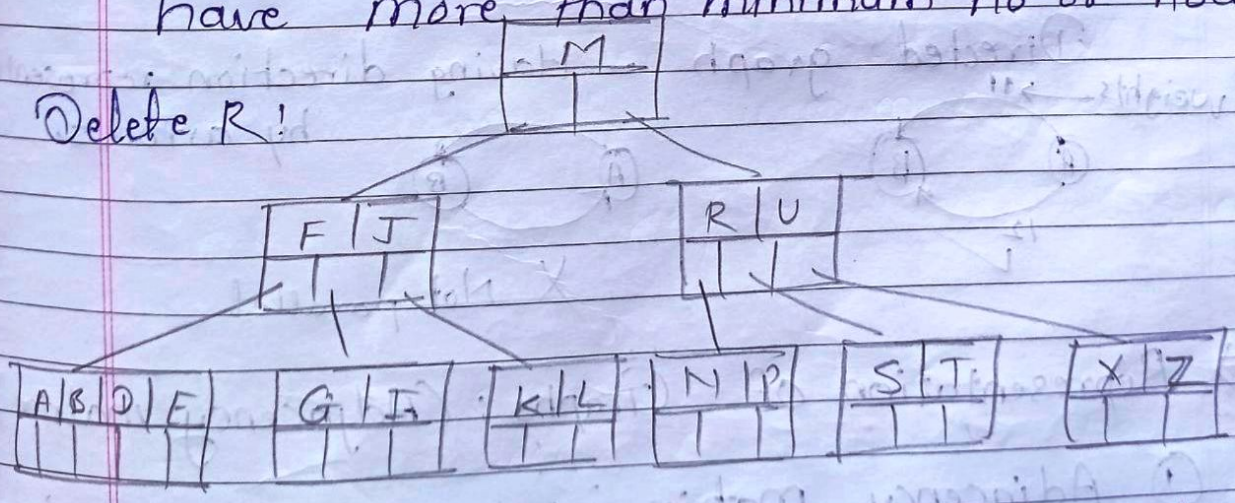
ABCDE
ABDE





Subcase E: inorder predecessor & successor and internal nodes & internal sibling not have more than minimum no of nodes.

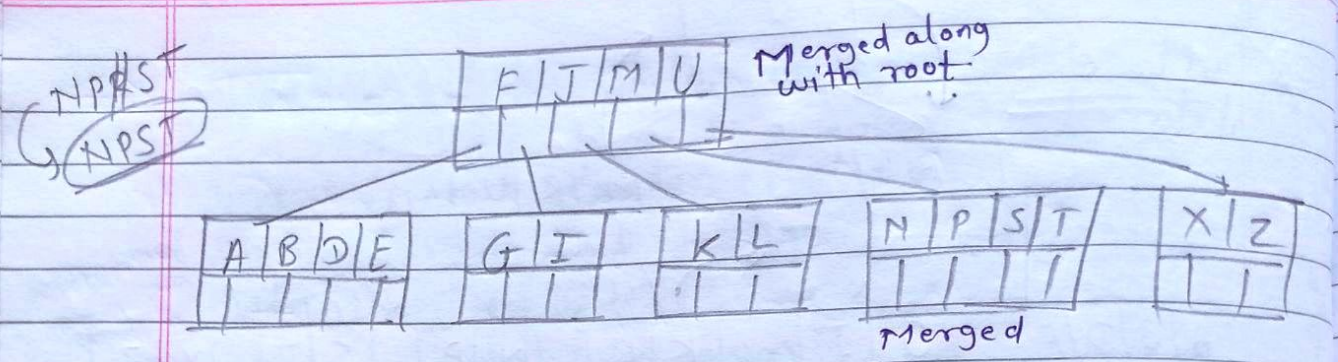
Delete R!



Solⁿ: Delete desired key merge inorder pred. & succ: merge internal nodes with root key.

Saathi

Date ___/___/___



lecture 27
29-10-22

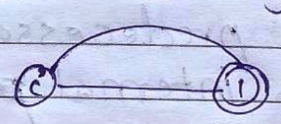
Unit 5

Graph.

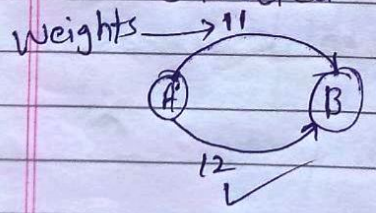
Graph is non-linear data structure it is collection of two sets namely set of vertices (V) & set of edges (E).

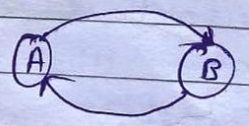
* Parallel edges

Undirected graph - Having no direction



Directed graph - Having direction represented by arrows.





X Not parallel.

⇒ Representation of Graphs : (Adjacency matrix)

① Adjacency matrix is a 2D array of size N X N where N is the number of vertices in a graph.

vertex – first node of graph
vertices – remaining node

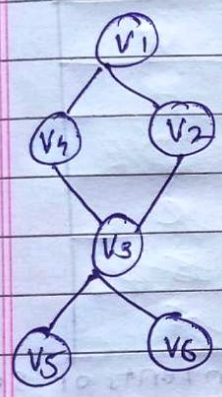
Date ___/___/_____

Saathi

Adjacency matrix for undirected graph is always symmetric

(2) Adjacency List: It is an array consisting of the address of all the linked lists. This representation can also be used to represent a weighted graph.

e.g

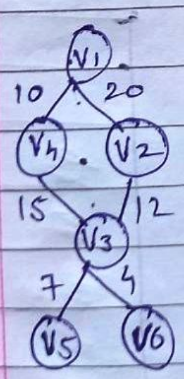


Array

V1	→ V4	→ V2			
V2	→ V1	→ V3			
V3	→ V4	→ V2	→ V5	→ V6	
V4	→ V1	→ V3			
V5	→ V3				
V6	→ V3				

we can also store weight

* ~~weights~~ with weights



V1	→ V2,20	→ V4,10			
V2	→ V3,12	→ V1,20			
V3	→ V2,12	→ V4,15	→ V5,7	→ V6,4	
V4	→ V1,10	→ V3,15			
V5	→ V3,7				
V6	→ V3,4				

Date ___/___/_____

⇒ Traversal of graphs:

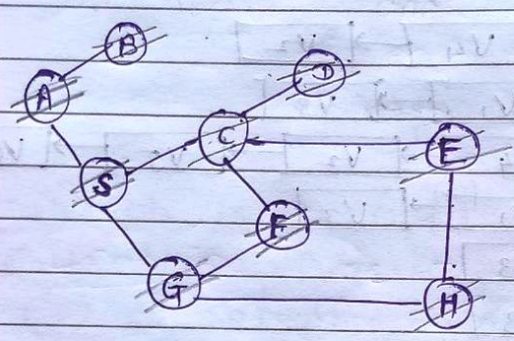
1) Depth first Search (DFS)
[Associated with Stack]

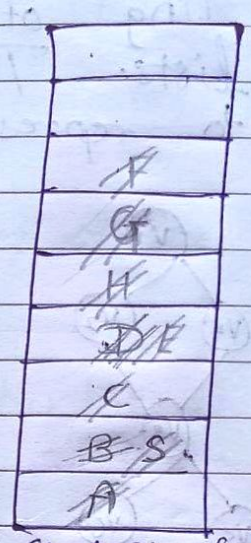
It is a data structure

push only one adjacent At at a time

Traverse in Alphabetical order (lexicographical)

Mark
↓
push
↓
output

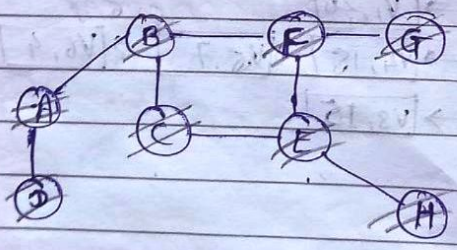




Contents of stack

DFS Traversal (output) | A, B, S, C, D, E, H, G, F |

e.g



Stack

DFS Traversal | A, B, C, E, F, G, H, D |

2) Breadth first search (BFS)

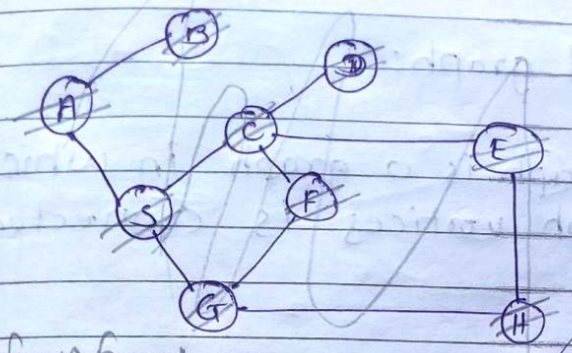
We use Queue data structure.

push all adjacent at one time

Date ___/___/_____

Saathi

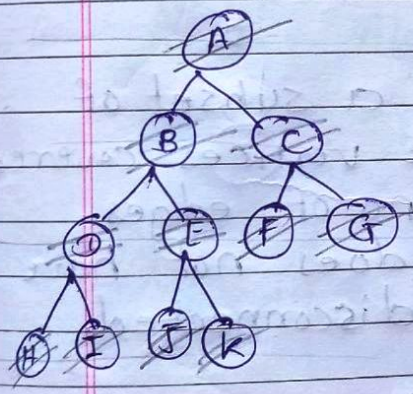
Traverse with adjacent node

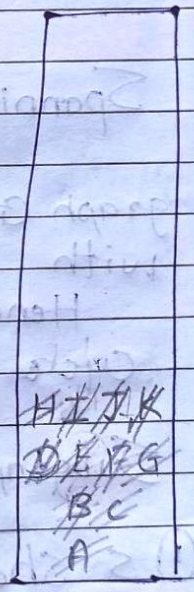


delete from front

Queue | ~~A~~ | ~~B~~ | ~~S~~ | ~~C~~ | ~~G~~ | ~~D~~ | ~~E~~ | ~~F~~ | ~~H~~ |

BFS Traversal [A, B, S, C, G, D, E, F, H]





DFS traversal [A, B, D, H, I, E, J, K, C, F, G]

Queue | ~~A~~ | ~~B~~ | ~~C~~ | ~~D~~ | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ | ~~I~~ | ~~J~~ | ~~K~~ |

BFS Traversal [A, B, C, D, E, F, G, H, I, J, K]

Page No.

Date ___/___/___

Complete undirected graph:

A complete graph is a graph in which each pair of the graph vertices is connected by an edge

Disconnected graph:

If at least two vertices of the graph are not connected by a path.

⟹ Spanning tree :-

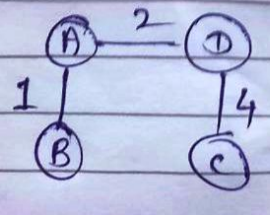
A spanning tree is a subset of graph G, which has all the vertices covered with minimum possible no of edges.

Hence, a spanning tree does not have cycle & it cannot be disconnected.

Subgraph 'S'. (Spanning tree)

1. S should be connected.
2. S should contain all vertices of G
3. S should contain (|V|-1) edges
4. S should not contain a cycle.

* weight of spanning tree.



weight of the above spanning tree is 1+2+4 = 7.

Date ___/___/_____

→ Minimum spanning tree.

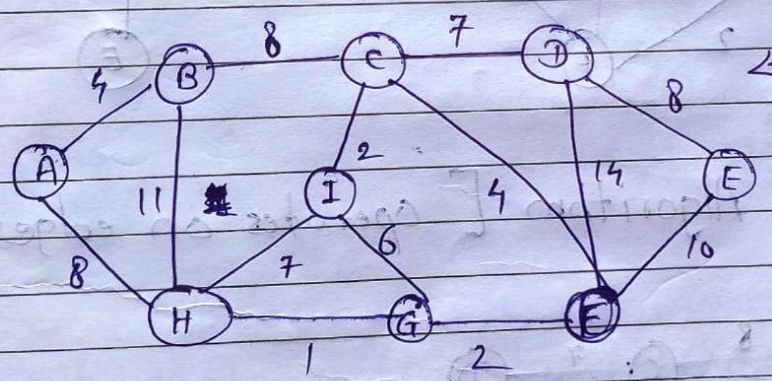
A minimum spanning tree (MST) is a spanning tree with the smallest possible weight among all the spanning trees for the given graph.

MST can be constructed using:

1) Kruskal's Algorithm : Uses edges
2) Prim's Algorithm : Uses Vertices.

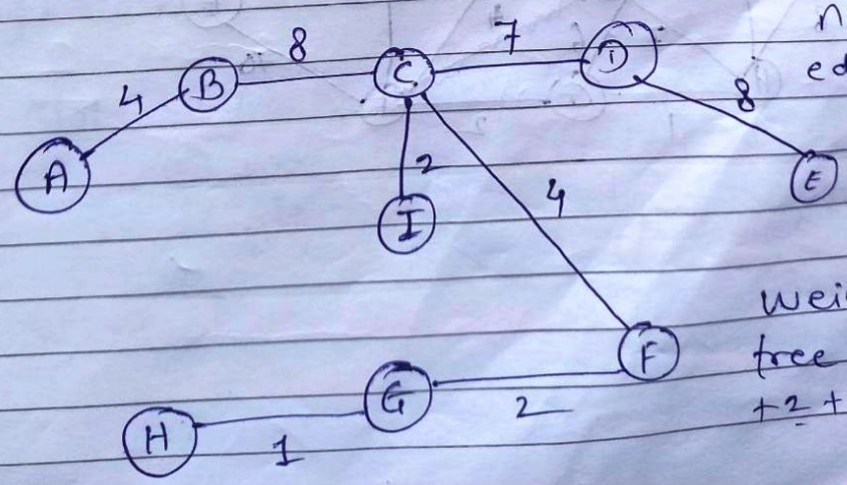
→ point → vertices

Prim's Algorithm



← undirected weighted graph

Constructing spanning tree using prim's algo.



n = 9 vertices

edges (n-1)

= 9 - 1 = 8

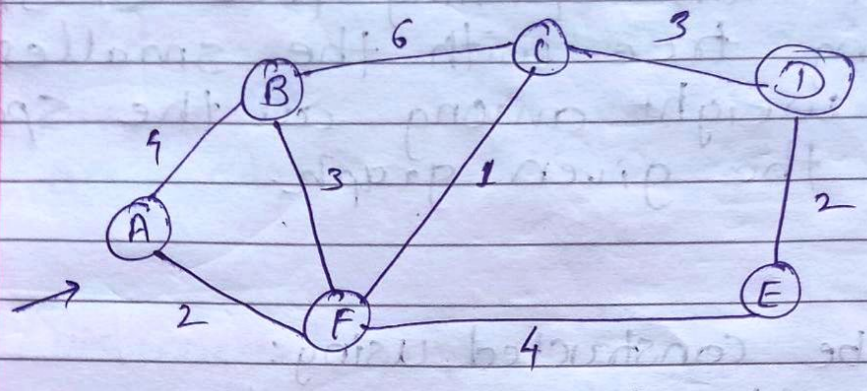
Weight of spanning tree 4 + 8 + 7 + 8 + 4 + 2 + 2 + 1 = 36

Page No.

Date ___/___/_____

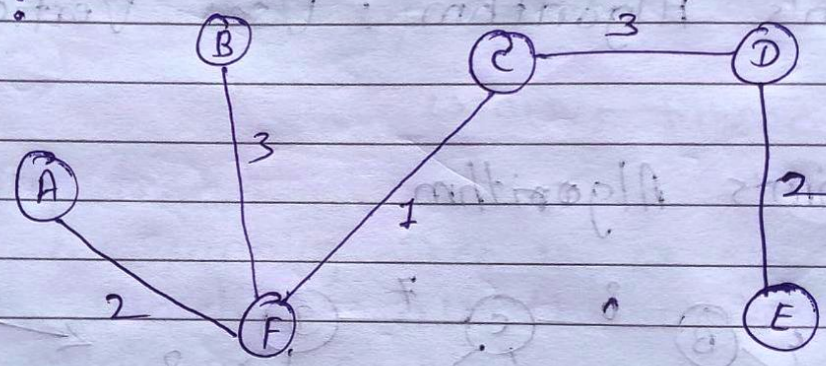
visited vertices : {A, B, C, F, G, H, D, E}

e.g

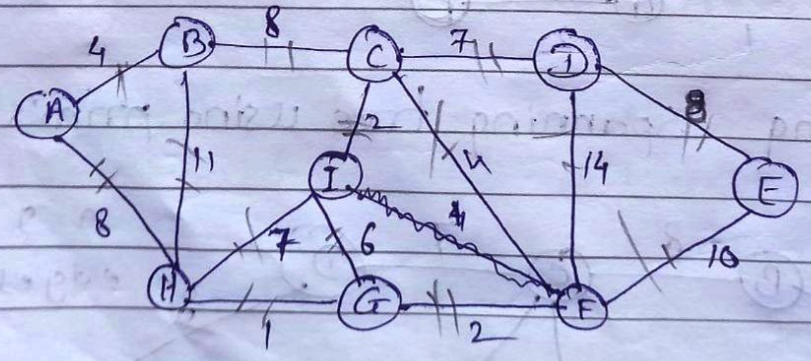


← graph.

MST :



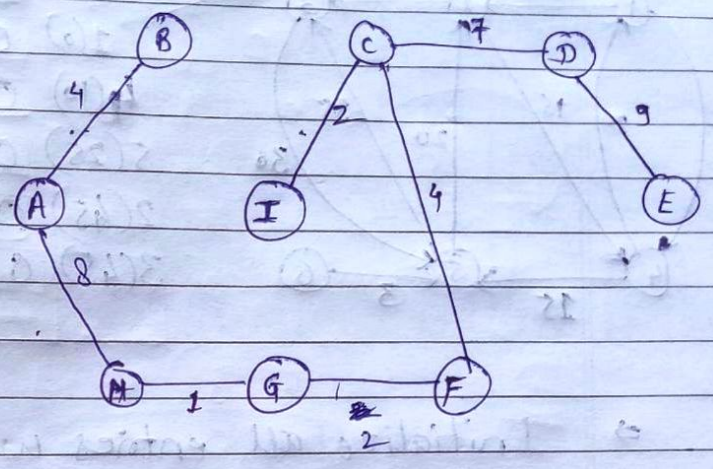
⟹ Kruskal's Algorithm [operates on edges]



Date ___/___/___

Arrange in ascending order of weights

✓ H-G	1
✓ C-I	2
✓ G-F	2
✓ A-B	4
✓ C-F	4
✗ I-G	6
✗ I-H	7
✓ C-D	7
✓ A-H	8
✗ B-C	8
✓ D-E	9
E-F	10
B-H	11
D-F	14



Stop if n-1 edges are added. otherwise continue.

Note: Prim's, Kruskals } Greedy algorithm.

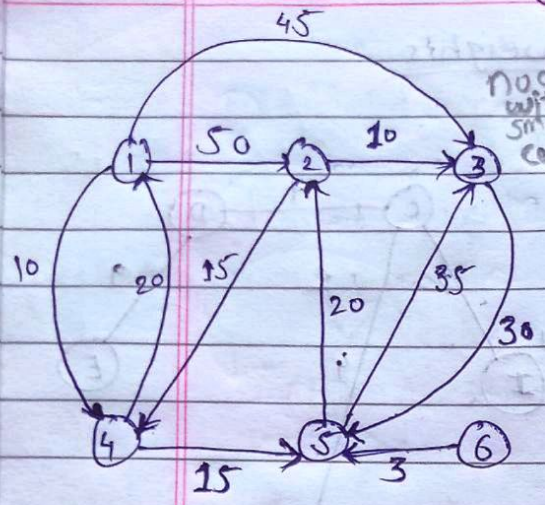
lecture 30
31-10-22

Shortest path algorithm

(1) Dijkstra Algorithm. (Greedy Algorithm)

Note - (1) Graph must be weighted for Dijkstra
(2) works on directed, undirected graph.
~~algorithm~~ ~~Greedy~~

Unit - 5



node with smallest cost

2 पर, 1 से आये

	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞
1(0)	0	50_1	45_1	10_1	∞	∞
4(10)	0	50_1	45_1	10_1	25_4	∞
5(25)	0	45_5	45_1	10_1	25_4	∞
2(45)	0	45_5	45_1	10_1	25_4	∞
3(45)	0	45_5	45_1	10_1	25_4	∞

⇒ Initialize all entries with ∞ & source node with 0

i j

if $(d(i) + c(i,j) < d[j])$

$d(1) + c(1,3) < \infty$ { 1 3 45

$d(1) + c(1,2) < \infty$ 1 2 50

$d(1) + c(1,4) < \infty$ 1 4 10 }

$d(j) = d(i) + c(i,j)$

prev[j] = i;

⇒ ~~Set~~ Find unselected node, make it current & mark it

1 - 2 = 45

1 - 3 = 45

1 - 4 = 10

1 - 5 = 25

1 - 6 = ∞

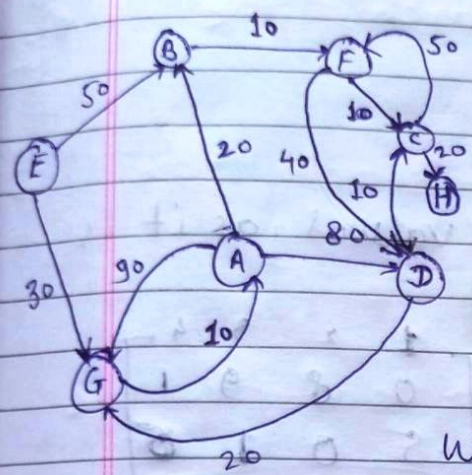
Paths

1 - 4 - 5 - 2 ←

prims
dikshra } Greedy floyd-war } dynamic

Date ___/___/___

e.g



	A	B	C	D	E	F	G	H
	1	2	3	4	5	6	7	8
	0	∞	∞	∞	∞	∞	∞	∞
A(0)	0	20_A	∞	80_A	∞	∞	90_A	∞
B(20)	0	20_A	∞	80_A	∞	B(20)+10=30 30_B	90_A	∞
F(30)	0	20_A	40_F	F(30)+40=70 70_D	∞	30_B	90_A	∞
C(40)	0	20_A	40_F	50_C	∞	30_B	90_A	60_C
D(50)	0	20_A	40_F	50_C	∞	30_B	70_D	60_C
H(60)							70_D	
G(70)	0	20_A	40_F	50_C	∞	30_B	70_D	60_C

Weights
A − B = 20
A − C = 40
A − D = 50
A − E = ∞
A − F = 30
A − G = 70
A − H = 60

path

A ← B ← F ← C ← H

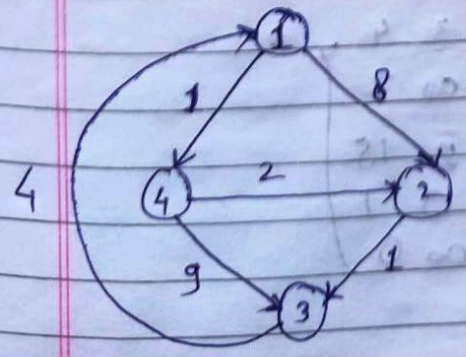
smallest path to ~~reach~~ reach H

A ← B ← F ← C ← D ← G

Small path to G

⇒ All pair shortest path (Floyd-Warshall Algorithm).

Works on positive or negative weights.
it uses dynamic approach.



$$D_0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & \infty & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & \infty & 0 & \infty \\ 4 & \infty & 2 & 9 & 0 \end{array}$$

$$D_1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & \infty & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 9 & 0 \end{array}$$

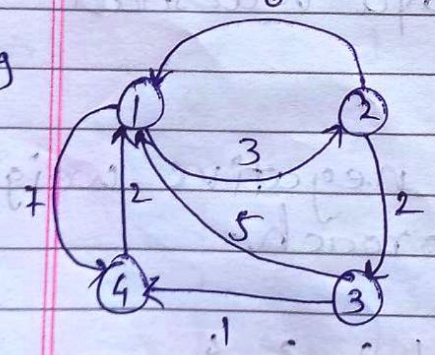
Infinity असलेला पूर्ण row or coloum चा values as it is लिहा. (using previous matrix)

$$D_2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 3 & 0 \end{array} \qquad D_3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & 5 & 0 & 1 & 5 \\ 3 & 4 & 12 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

$$D_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 4 & 1 \\ 2 & 5 & 0 & 1 & 3 \\ 3 & 4 & 7 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

Final matrix.

e.g



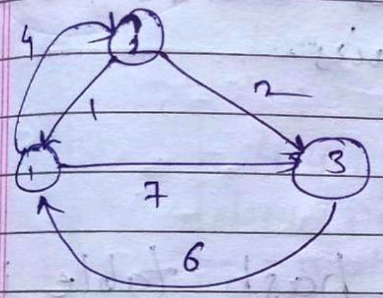
$$D_0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & \infty \\ 3 & 5 & \infty & 0 & 1 \\ 4 & 2 & \infty & \infty & 0 \end{array}$$

$$D_1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array}$$

$$D_2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array} \qquad D_3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 7 & 0 & 2 & 3 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

$$D_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 5 & 0 & 2 & 3 \\ 3 & 3 & 6 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

e.g.



$$D_0 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 7 \\ 2 & 1 & 0 & 2 \\ 3 & 6 & \infty & 0 \end{array}$$

$$D_1 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 7 \\ 2 & 1 & 0 & 2 \\ 3 & 6 & 10 & 0 \end{array} \qquad D_2 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 1 & 0 & 2 \\ 3 & 6 & 10 & 0 \end{array}$$

$$D_3 = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 1 & 0 & 2 \\ 3 & 6 & 10 & 0 \end{array}$$ ~~final~~ final.

lecture32
2-11-22

Unit 6

Hash table, Searching & Sorting.

Can we reduce the search time lower than $O(n)$, $O(\log n)$?

Yes, we can reduce the seach time lower than $O(n)$, $O(\log n)$. By making use of use of Hash Tables!

Hash table is a type of datastructure that stores key-value pairs.

★ Load factor :

Load factor (λ) of a hash table is the ratio of the number of elements in the hash table to the table size.

$$\lambda = \frac{\text{(no. of elements populated in hash table)}}{\text{(size of the hash table)}}$$

Note : ① The size should be chosen such that the load factor is less than 1

② Table capicity is usually chosen to be a prime number ~~to spread out~~

⇒ Types of hashing functions.

① Division method.

key - actual no.
value - Address or key of actual no.

Date ___/___/___

The hashing function $h(k)$ can be written as:
$h(k) = k \% M$ where, M is size of table.

② Mid Square Method

Suppose hash table has 100 memory locations, so $r=2$ because two digits are required to map the key to memory location.

if 1000 memory loc. then $r=3$.

k	4147	3750	2103
k^2	~~17197609~~	~~14062500~~	~~4422609~~
H(k)	97	62	22

③ Multiplication method.

1) Multiply the key k(50) by a constant A(0.485), where $0 < A < 1$

2) Extract the fractional part of KA

3) Multiply the fractional part by M

4) Truncate the result to get the result in the range 0 to (M-1)

Formula: $h(k) = \lfloor M (kA \bmod 1) \rfloor$

$= \lfloor 10 (50 \times 0.485 \bmod 1) \rfloor$

Page No.

Date ___/___/___

$= \lfloor 10\,(24.25 \bmod 1) \rfloor$

$= \lfloor 10\,(0.25) \rfloor$

$= \lfloor 2.5 \rfloor = \underline{\underline{2}}$

(4) Folding: A) Fold Shift

Suppose we have table size of 100 i.e hash table with 2 digit index (last part can have less digit)

K = key's

k	2103	7148	12345
k_1, k_2, k_3	21, 03	71, 48	12, 34, 5
$h(k) = k_1 + k_2 + k_3$	h(2103) = 21+03 = 24	h(7148) = 71+48 = 119	h(12345) = 12+34+5 = 51

B) Fold ~~boundary~~ boundary.

12 ⋮ 34

↓ folding

12
43

Note: if r=2 then we will get key's ranging between $00 \leq k \leq 99$

k	2103	7148	12345
k_1, k_2, k_3, k_4	21, 03	71, 48	12, 3, 4, 5
Reversing k_2, k_4	21, 30	71, 84	12, 43, 5
$h(k) = k_1 + k_2 + k_3$	h(2103) = 21+30 = 51	h(7148) = 71+48 = ~~119~~ 55	h(12345) = 12+34+5 = 60

Suppose k = 1 2 3 4 5 6 7 8 9 1 (2 Even, 4 Even) → 12, 43, 56, 87, 91

12+43+56+87+1 = ~~189~~ = 89

89 is key

above no will

Date ___/___/_____

Saathi

⇒ Collision

When same value is generated for two different key's (no) then collision has occured

Collision resolution Techniques.

- Separate Chaining (open hashing)
- Open Addressing (closed Hashing)
  - Linear probing
  - Quadratic probing
  - Double probing

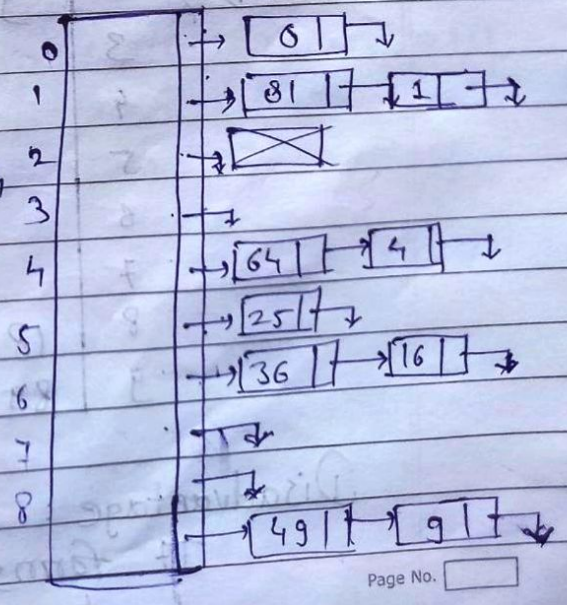
1) Separate chaining.

In this technique, linked lists are associated with each bucket of the hash table.

Concider e.g

$h(k) = k \% 10$

49, 36, 25, 16, 9, 64, 0, 81, 1, 4



Page No.

Separate chaining is used when we normally do not know in advance the total number of key/ records.

2) Open Addressing:

Open addressing used when we know the no. of keys/ records in advance

Table size should be at least two times that of the number of keys to be inserted.

* A) Linear Probing $f(i) = 1, 2, 3, \ldots$

e.g. Concider the following data & hash fun?

$h(k) = k \% 10$ (Table size = 10)

89, 18, 49, 58 & 9

f(1) =>	0	49
f(3) =>	1	58
f(4) =>	2	9
	3	
	4	
	5	
	6	
	7	
	8	18
	9	89

In linear probing F is linear function of $f(i) = i$

Linear probing attempts to find an empty cell in case of collision.

The value is increased by 1 after every non-empty cell is encountered

Disadvantage: (Primary cluster)

It forms clusters that makes values harder to find.

Date ___/___/_____

Saathi

* B) Quadratic probing. $f(i^2) = 1^2, 2^2, 3^2 \ldots$

e.g consider the data & hash function $h(k) = k \cdot 1.7$
(Table size = 7)

76, 40, 48, 5 and 55, 62

$76 \% 7 = 6$	0	48	$\Leftarrow f(1^2) = 1$
$40 \cdot 1 \cdot 7 = 5$	1	62	$\Leftarrow f(3^2) = 9$
$48 \cdot 1 \cdot 7 = 6$	2	5	$\Leftarrow f(2^2) = 4$
$5 \cdot 1 \cdot 7 = 5$	3	55	$\Leftarrow f(2^2) = 55$
$55 \cdot 1 \cdot 7 = 6$	4		
$62 \cdot 1 \cdot 7 = 6$	5	40	
	6	76	

Disadvantage :
It eliminates primary clustering but impose secondary clustering. A secondary clustering will develop & grow in size

* c) Double Hashing.

Double hashing eliminates secondary clustering, by having two hash fun^n that both depend on the hash key.

Page No.

Date ___/___/___

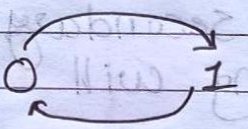
Concider the data 89, 18, 49, 58, 69, size=10

& hash functions $h_1(k) = k \% 10$ $h_2(k) = 7 - (k \% 7)$

0	69	⇐ $7-(69\%7) = 1$ from 9th pos.
1		
2		
3	58	⇐ $7-(58\%7) = 5$ from 8th pos.
4		
5		
6	49	⇐ $7-(49\%7) = 7$ from 9th position
7		
8	18	
9	89	

lecture 33
2-11-22

⇒ Rehashing.



We know that as load factor tends towards the one, the performance of hash table becomes slower, So whenever the load factor reaches to a certain value i.e between 0.70 to 0.75, it is appropriate to resize hash table to a larger size, and rehash all the data from the old table to the newer table.

The size of new table should be at least twice the size of the current table or the next prime number.

Page No.

e.g of rehashing 13, 15, 6, 24 hashing with linear probing

$h(x) = x \% 7$

$\lambda = 0.57$

0	6
1	15
2	
3	24
4	
5	
6	13

$h(x) = x \% 17$

$\lambda = 0.29$

size $7 \times 2 = 14$

↓ prime nos.

15 ✗

16 ✗

17 ✓

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Insert 23

$\lambda = 0.71$

0	6
1	15
2	23
3	24
4	
5	
6	13

⟹ rehashing

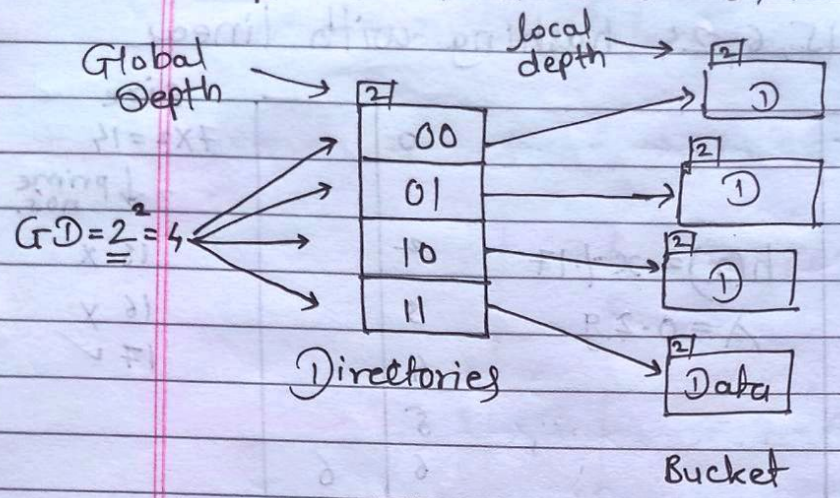
⇒ Extendible Hashing.

It is a dynamic hashing where in directories, and buckets are used to hash data.

1) dynamic — hash funⁿ, table size — keeps changing.

Date ___/___/_____

Basic Structure of Extendible Hashing.



Global depth denotes the bits

$GD \geq LD$

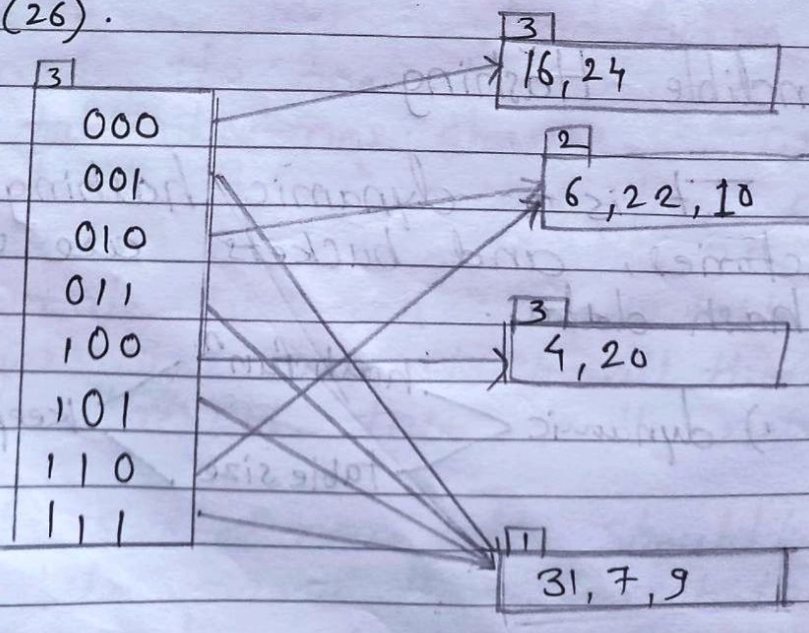
if

GD == LD ⇒ expand directories

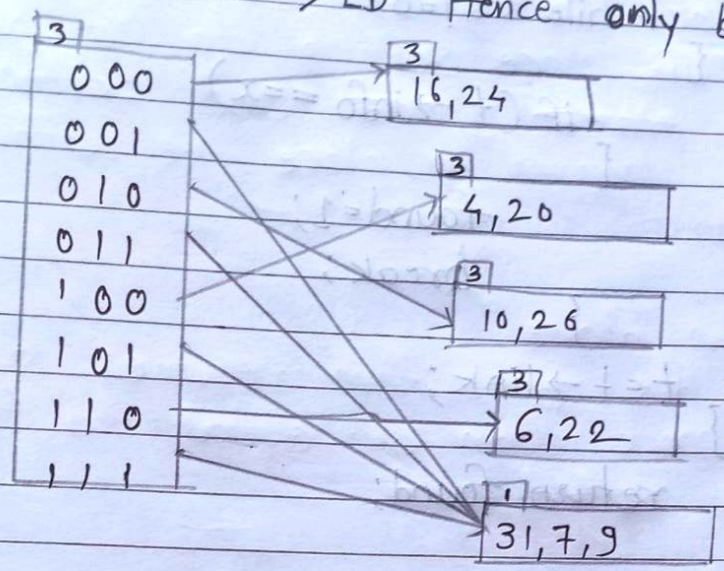
GD > LD ⇒ split buckets

In overflow condition directories are expands
Buckets are splits.

e.g 10000 (16), 00100 (4), 00110 (6), 10110 (22), 11000 (24), 01010 (10), 11111 (31), 00111 (7), 01001 (9), 10100 (20), 11010 (26).



Insert 26 GD > LD Hence only bucket will split.



जितना LD उतने digits consider करने है if LD=3 then 000, 001, 111
LD=2 00, 01, 11

⇒ ~~Linear~~ Searching:

1) Linear search (Array in a Linked List)

```
int Linear_Search(int x)
{
    int i;
    for( i=0; i<n; i++)
        if (arr[i]==x)
            return i;
        return -1;
}
```

(2) Linear search (Linked List)

```
int Search (int x)
{
    struct node *t=head;
    int found=0;
```

Date ___/___/___

```
        while (t!=0)
        {
            if (t->info==x)
            {
                found=1;
                break;
            }
        t=t->link;
        }
        return found;
    }
```

② Linear search (singly circular link list)
③ Linear search (Doubly Link list)

2] Binary search (In array)

```
int binarySearch(int arr[], int l, int r, int x)
{
    while (l<=r)
    {
        int m=l+(r-l)/2;

        if (arr[m]==x)
            return m;
        if (arr[m]<x)
            l=m+1;
        else
            r=m-1;
    }
```

0	1	2	3	4	5	6	7
10	20	38	40	45	90	101	111

$l = 0$
$r = 7$

2) B Binary search (In tree)

```
int search (struct bstnode *nroot, int x)
{
        if (nroot == NULL)
                return 0;
    if (nroot -> data == x)
                return 1;
    else  if (x > nroot -> data)
                return search(nroot->rchild, x);
    else
                return search (nroot -> lchild, x);
}
```

lecture 35
4-11-22

⇒ Sorting Techniques.

1) Internal sorting : If all the data that is to be sorted can be adjusted at a time in main memory, then internal sorting method is being performed

2) External sorting : If all the data that is to be sorted canotbe adjusted at a time in the main memory, then External sorting is being performed.

Date ___/___/___

1) In place Sorting
In place sorting do not require any extra ~~place~~ space and sorting said to be happen in place i.e within the array itself.

2) Not in place
This algorithm requires space which is more than or equal to the elements being sorted.

1] Bubble sort technique.

Unsorted

e.g 22, 35, 17, 5, 85, 66, 57, 48, 95, 16

~~22, 35, 5, 17, 66, 85, 57, 48, 85, 16, 95~~

~~22, 5, 17, 35, 57~~

1 22, 17, 5, 35, 66, 57, 48, 85, 16, 95
2 17, 5, 22, 35, 57, 48, 66, 16, 85, 95
3 5, 17, 22, 35, 48, 57, 16, 66, 85, 95
4 5, 17, 22, 35, 48, 16, 57, 66, 85, 95
5 5, 17, 22, 35, 16, 48, 57, 66, 85, 95
6 5, 17, 22, 16, 35, 48, 57, 66, 85, 95
7 5, 17, 16, 22, 35, 48, 57, 66, 85, 95
8 5, 16, 17, 22, 35, 48, 57, 66, 85, 95
9 5, 16, 17, 22, 35, 48, 57, 66, 85, 95

If there are n numbers then total passes are $\underline{\underline{n-1}}$

Page No.

ceil [5.5] ↑ = 6
floor [5.5] ↓ = 5

Date ___/___/___

Saathi

lecture 38

left child = root * 2+1
right child = root * 2+2

How to know that there is a subtree below a given node?

root * 2+1 <= ub

How to know that ~~there is a subtree below~~ a given node has only left child?

root * 2+1 == ub

Max Heap ⇒ Ascending order
Min Heap ⇒ Descending order.

```
void heapsort (int numbers[], int ub)
{
        int i, temp;
                                Last elem ka parent ka formula
Creating heap   for(i= (int) ceil((ub/2.0)) - 1; i>=0; i--)
                {
                        siftDown(numbers, i, ub);
                }
Sorting heap    for( i= ub; i>=1; i--)
                {
                        temp = numbers[0];
                        numbers[0] = numbers[i];
```

Page No.

Heapify म्हणजेच heap tree चा properties satify करणे e.g max heap parent > child

Date ___/___/___

```
        numbers[i] = temp;
        siftDown(numbers, 0, i-1);
    }
}

void siftDown(int numbers[], int root, int bottom)
{
        int done, maxChild, temp;
        done = 0;
        while((root*2+1 <= bottom) && (!done))
        {
            if (root*2+1 == bottom ||
            numbers[root*2+1] > numbers[root*2+2])
                maxchild = root*2+1;
            else
                maxchild = root*2+2;
        if (numbers[root] < numbers[maxChild])
        {
            temp = numbers[root];
            numbers[root] = numbers[maxChild];
            numbers[maxChild] = temp;
            root = maxChild;
        }
        else
            done = 1;
    }
}
```

finding maxchild

swapping if root < maxchild