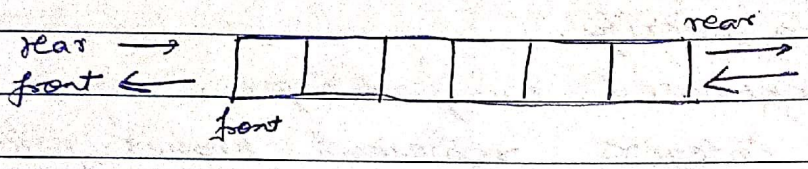
Date 26/10/21

Lecture – 11

Implementation of priority queue using multilevel queue :-

Double ended queue :-

Insertions and deletions can be done at both ends of the queue.



Linked Linear List (Linked List) :-

Static memory allocation

↳ memory allocated at compile time.

```
int x;
x = 10;
```

Dynamic memory allocation

↳ memory allocated at run (execution) time.

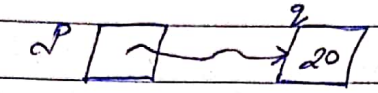
e.g. malloc f^n in C

e.g

```
struct node {
        int roll;
        float cgpa;
};
struct node *ptr;
```

e.g. malloc function in C

```
int *p;
int q;
p = &q;
*p = 20;
```



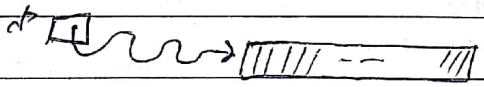
q refers to the contents of q by *p

```
int *p;
p = (int *) malloc (sizeof(int));
```

```
typecasting
char *
scanf ("%d", p);
*p = *p + 1;
```

portability

```
int *p;
int n;
printf ("How many?");
scanf ("%d", &n);
p = (int *) malloc (sizeof(int) * n);
p[0] = 50;
```



Linked List :-

It is a liked representation of a linear list.

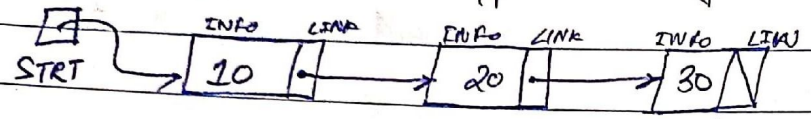
e.g. (10 20 30)

We have node's that are dynamically allocated.

A node is, a block of memory. Each such block has an address.

Each node has two parts: (fields) → information part, → link part

Link field is a pointer field.



Date
20/10/2021

Lecture - 12

In order to access the content (fields) of a node, we must have its address.

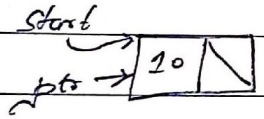
- In C language, each node can be declared as a structure.
- Self-referential.

```
struct node {
    int info;
    struct node *link;
};
struct node *ptr, *start, *save;
ptr = (struct node *) malloc (sizeof (struct node));
start = ptr;
save = ptr;
```

```
ptr -> info = 10;
ptr -> link = null;

ptr = (struct node *) malloc (sizeof (struct node));
ptr -> info = 20;
ptr -> link = null;
save -> link = ptr;
save = ptr;

ptr = (struct node *) malloc (sizeof (struct node));
ptr -> info = 30;
ptr -> link = null;
save -> link = ptr;
save = ptr;
```

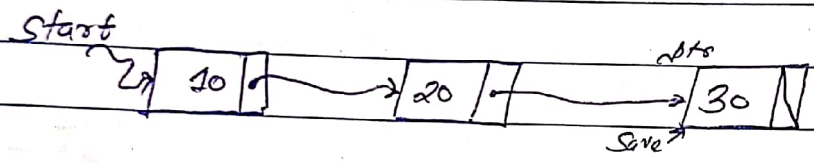
Date
23/10/2021

Lecture - 13




```
int x;
printf ("Enter information : ");
scanf ("%d", &x);
ptr = (struct node *) malloc (size of (struct node));
ptr -> info = x;
ptr -> link = null;
start = ptr;

printf ("Continue ?");
scanf ("%d", &choice);
```

```
while (choice) {
    printf ("Information: ");
    scanf ("%d", &x);
    ptr = (struct node *) malloc (sizeof(struct node));
    ptr → info = x;
    ptr → link = null;
    save → link = ptr;
    save = ptr;
    printf ("Continue?");
    scanf ("%d", &choice);
}
```



* Create a singly linked linear list where each node stores an integer. The entry stops when -999 is entered.

Traversing the list :-

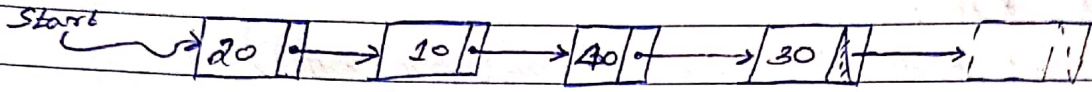
```
trav = start;
while (trav != null) {
    printf ("%d\n", trav → info);
    trav = trav → link;
}
```

Lecture – 14

Date
25/10/2021

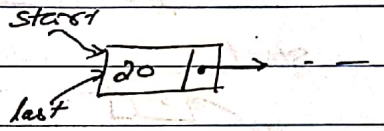
Insertion of nodes in a linked list :-

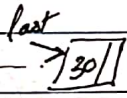
1) Insertion at the end :-



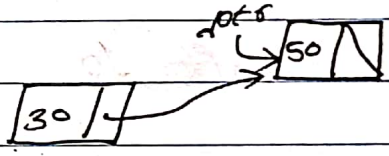
How many addresses to be found?
Which pointers are to be modified?

```
struct node *last;
last = start;
while (last -> link != null)
    last = last -> link.
```





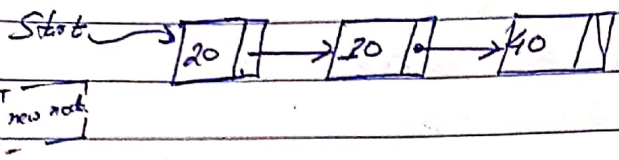

```
ptr = (struct node *) malloc (sizeof (struct node));
printf ("Enter new element: ");
scanf ("%d", &ptr -> info);
ptr -> link = null;
last -> link = ptr
```



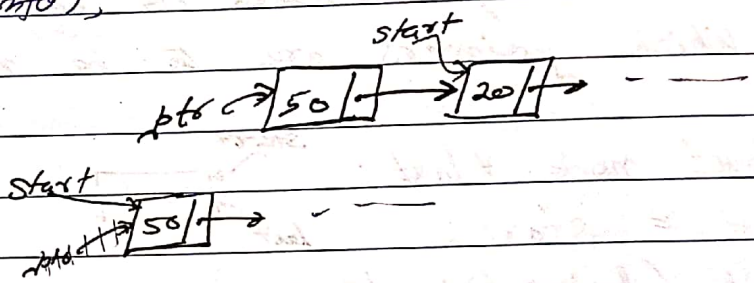
Handling of empty list :-

```
if (start == null) {
    ptr = - -
    ptr -> info = ——
    ptr -> link = null;
    start = ptr; }
else { - - - }
```

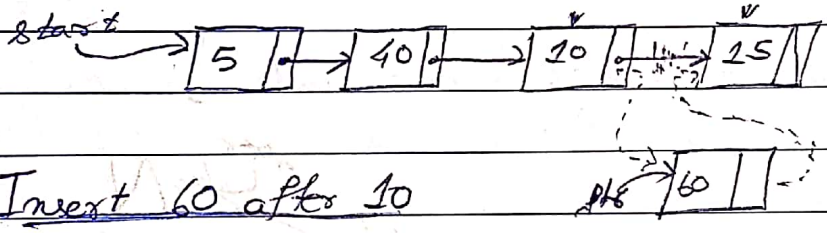
2> Insertion at the beginning of a list :-




```
struct node *ptr;
ptr = (struct node *) malloc (sizeof (struct node));
scanf ("%d", &ptr->info);
ptr->link = start;
start = ptr;
ptr = null;
```

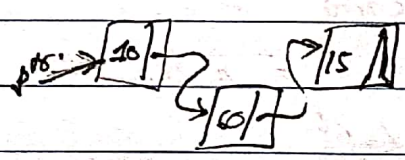


2> Insertion after a given node :



Insert 60 after 10

```
struct node *trav, *next;
trav = start;

while (trav->info != 10)
    trav = trav->link;
ptr = (struct node *) malloc (size of (struct node));
ptr->info = 80;
next = trav->link;
trav->link = ptr;
ptr->link = next;

trav = ptr = next = null;
```



Date
27/10/2021

Lecture -

Insertion :-

Applications of stack :-

palindromes

a b b a

malayalam

Balanced Parenthesis

✓ (((())))

✗ (((()

✓ Asymptotic Complexity

✓ Stack

✓ Queue

Insertion :

```
if (start == null) {      }
else {      }
```

Element not present in the list :

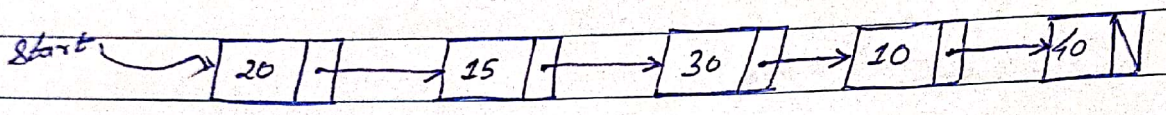
```
:
:
while (trav -> info != x)
{
    trav = trav -> link;
    if (trav == null)
        break;
}
if (trav != null) { ---- }
else { printf ("Element not found!");
```

```
:
while (trav -> info != x && trav != null)
    { trav = trav -> link; }
if (trav != null) { ... }
else { ... }
```

Short circuiting

Deletion of element :-

1) Deletion of a particular element.



Delete 30 from the list :-

```
int x;
struct node *trav, *save = null;
printf ("Which element to delete ?");
scanf ("%d", &x);

trav = start;
while (trav -> info != x) {
        save = trav;
        trav = trav -> link;
    }

save -> link = trav -> link;
trav -> link = null;
free (trav);
```

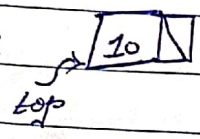
Lecture - 16

Date
30/10/21

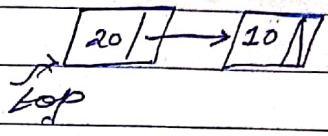
Implementation of stack using linked linear list :-

Stack has LIFO property.
e.g. push 10, push 20, push 30

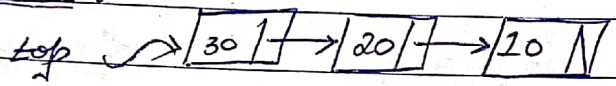
After push 10 :



After push 20 :



After push 30 :



Code :-

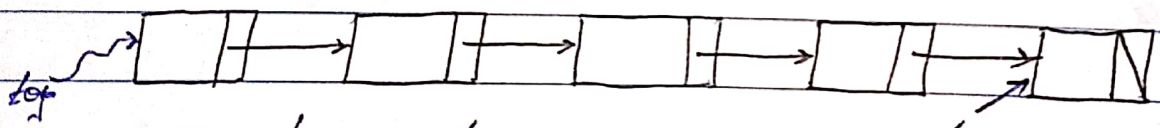
```
Struct node {
        int info;
        struct node *link;
};
Struct node *top, *ptr, *temp;
top = null;
int main() {
    top = null;
    :
    :
}
```

```
void push() {
    int x;
    printf("Enter information part: ");
    scanf("%d", &x);
    ptr = (struct node *) malloc(size of (struct node));
    ptr -> info = x;
```

```
ptr -> link = null;
if (top == null)
        top = ptr;
else
    { ptr -> link = top;
      top = ptr;
    }
}
```

```
void pop () {
    if (top == null)
    {
        printf ("Stack is empty");
        return;
    }
    temp = top;
    top = top -> link;
    temp -> link = null;
    free (temp);
}
```

O(1)




```
trav = top;
while (trav -> link != null)
        trav = trav -> link;
```

O(N)

Lecture-17

Date
01/11/2021

#Implementation of queue using linked list :-

Two pointers variables : front, rear

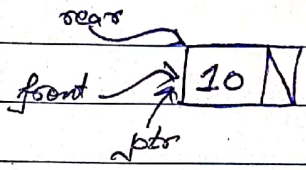
→ struct node *front, *rear;

Initially,

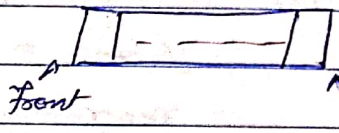
front = null;

rear = null;

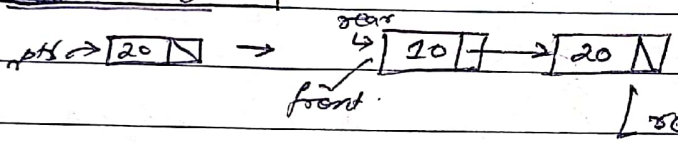
Insert 10 :-



for queue,



Insert 20

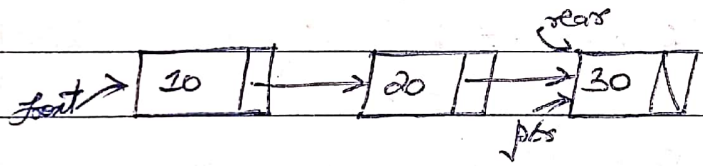


[rear must point to new inser

∴ rear → link = ptr;

rear = ptr;

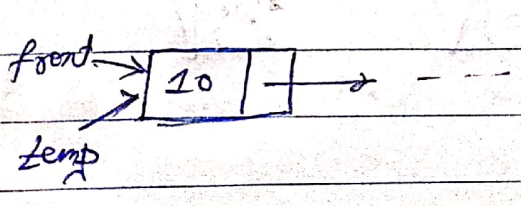
Insert 30 :-



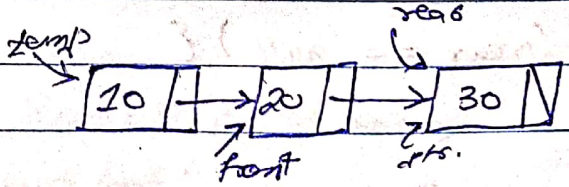
rear → link = ptr;

rear = ptr;

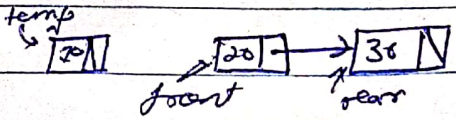
Delete :-



Then,



Then,

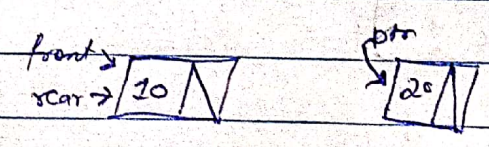


[free(

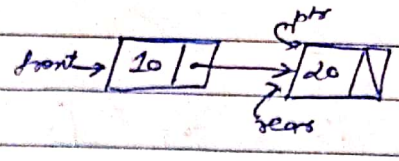
(32)

Code :-

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

Struct node *front; *rear, *temp, *ptr;

int main() {
        front = NULL;
        rear = NULL;
        enqueue();
        enqueue();
        dequeue();
        return 0;
}

void enqueue() {
        int x;
        printf("Enter element to insert : ");
        scanf("%d", &x);

        ptr = (struct node *) malloc (sizeof(struct node));

        ptr -> info = x;
        ptr -> link = null;
        if (rear == null) {
                front = ptr;
                rear = ptr;
        }
```



```
else {
    rear -> link = ptr;
    rear = ptr.
}
}
```




```
void dequeue() {
    (!!) else {
        temp = front;
        front = front -> link;
        temp -> link = null;
        free(temp);
    }
}
```

(!!) → Insert (missing)

```
if (front == null)
{
    printf("Queue Under
    return;
}
```

```
void display() {
    temp = front;
    while (temp -> link != null) {
        printf("%d\n", temp -> info);
        temp = temp -> link;
```

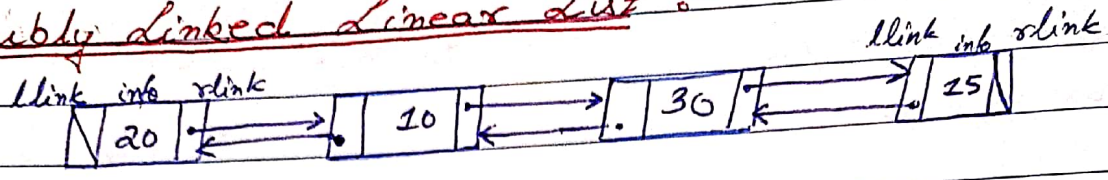
inside while loop

```
void display() {
    temp = front;
    while (temp != null) {
        printf("%d\n", temp -> info);
        temp = temp -> link;
    }
}
```

```
for (temp = front; temp != null; temp = temp -> link) {
    printf("%d\n", temp -> info);
}
```

Lecture - 18

Doubly Linked Linear List :-

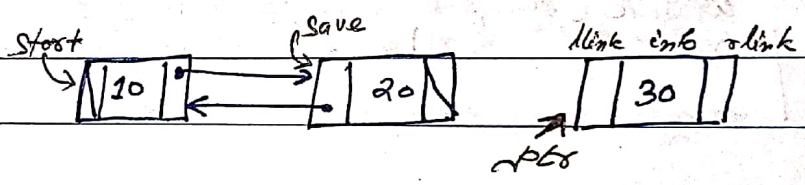


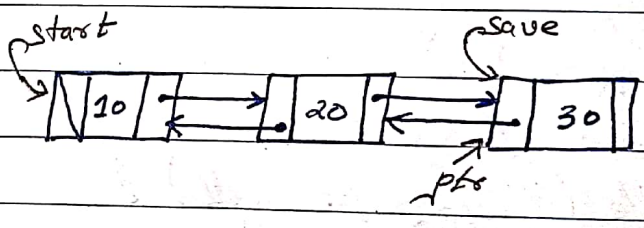

```
Struct node {
        int   info;
        struct node *rlink;
        Struct node *llink;
}
```

Creation of a doubly linked list :-

Initially start = null.

```
ptr = (struct node *) malloc(sizeof(struct node));
ptr → info = 10;
ptr → llink = null;
ptr → rlink = null;
Start = ptr;
Save = ptr;
ptr → info = 20;
Save → rlink = ptr;
ptr → llink = Save;
Save = ptr;
ptr → info = 30;
```

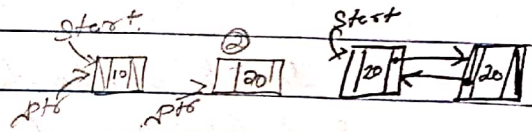




Date
08/12/2000

Lecture - 19

Implementation :-




```
struct node {
    int info;
    struct node *rlink;
    struct node *llink;
};
struct node *start;

int main() {
    :
    create();
    :
    :
    return 0;
}

void create() {
    int element, choic=1;
    struct node *ptr, *save;

    printf("Enter information : ");
    scanf("%d", &element);

    ptr = (struct node *)malloc(sizeof(struct node));
    ptr -> info = element;
    ptr -> rlink = null;
    ptr -> llink = null;
①-  start = ptr;
    save = ptr;
    printf("Do you want to continue? 1-yes, 0-No");
    scanf("%d", &choice);

while(choice) {
    printf("Enter information : ");
    scanf("%d", &element);
②-  ptr = (struct node *) malloc(sizeof(
        struct node));
    ptr -> info = element;
    ptr -> rlink = null;
    ptr -> llink = null;
    save -> rlink = ptr;
    ptr -> llink = save;
    save = ptr;
    printf("Do you want to continue");
    scanf("%d", &choice);
}
}
```

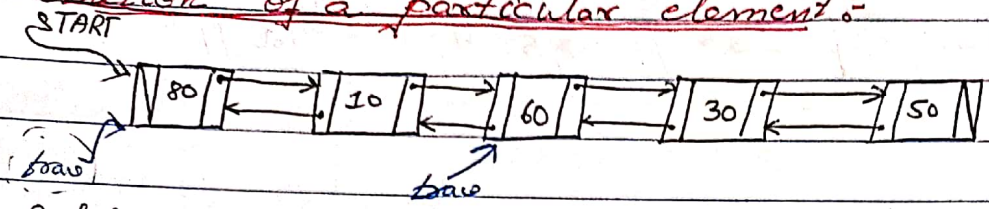
Q. Create a doubly linked list always by inserting new node at beginning of the list.

Insertion of a node after a given node.


```
void insert_after() {
    int x, y
    struct node * trav;
    printf("After which ?");
    scanf("%d", &x);
    printf("New element ?");
    scanf("%d", &y);
    trav = start;
    while (trav->info != x)
        trav = trav->rlink;
    ptr = (struct node *) malloc (size of (struct node));
    ptr->info = y;
    ptr->rlink = trave->r link
    ptr->llink = trav;
    trav->rlink->llink = ptr;
    trav->rlink = ptr;
```

Lecture - 20

Deletion of a particular element :-

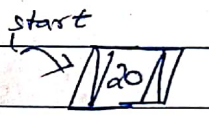


Delete 60

```
trav -> llink -> rlink = trav -> rlink;
trav -> rlink -> link = trav -> link;
trav -> rlink = null;
trav -> llink = null;
trav = null;
free(trav);
```

```
# if (start == NULL) {
  }
  else {
  }
```

Also, if

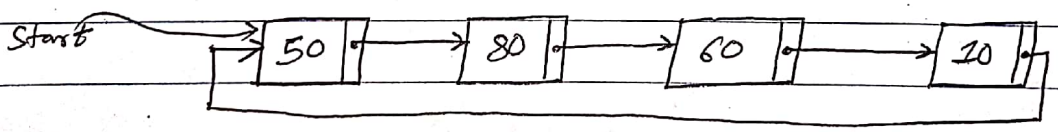


i.e. single node

```
trav = start;
while (trav -> info != 20)
    trav = trav -> rlink
```

Singly Linked Circular List :-

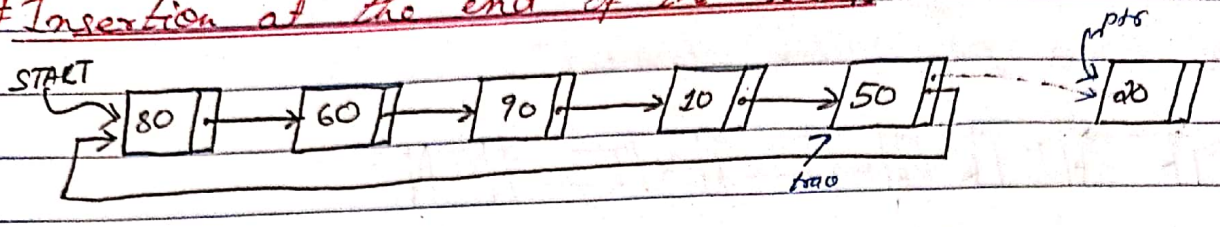
(Circular Singly Linked List)



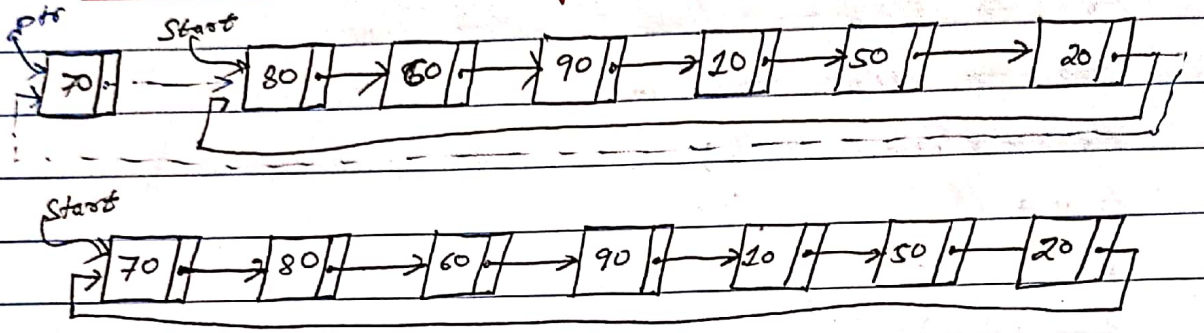

```
trav = start;
printf("%d \n", trav -> info);
trav = trav -> link;
while (trav != start) {
    printf("%d \n", trav -> info);
    trav = trav -> link;
}
```

```
while (trav != null)
{
    printf("%d \n", trav -> info);
    trav = trav -> link;
}
```

Insertion at the end of the list :-



Insertion at the beginning of list :-



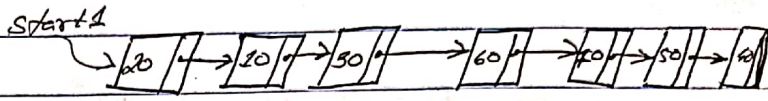
Q. Create a singly linked circular list
(i) always by inserting at end.
(ii) always by insertion at beginning.

Date
15/11/2021

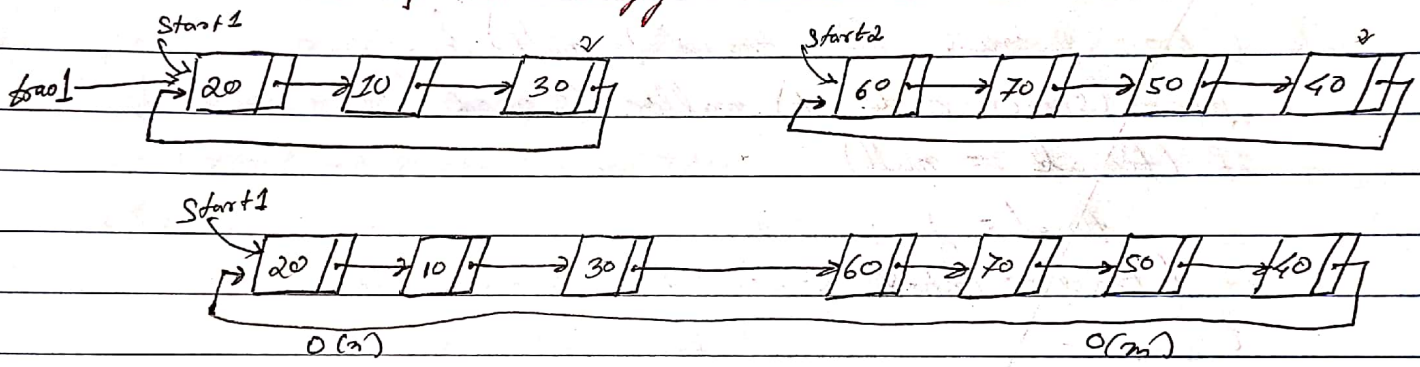
Lecture-21

#Concatenation of two singly linked non-circular lists :-


```
trav = start1;
while (trav -> link != null)
    trav = trav -> link;
trav -> link = start2;
trav = start2 = null;
```



Concatenation of two singly linked circular lists :-



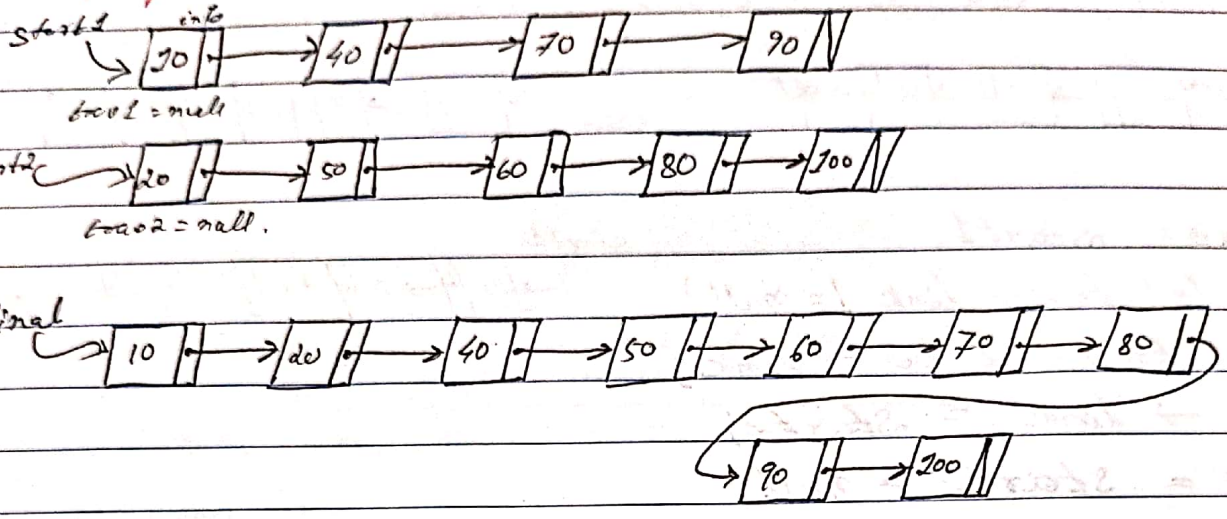

```
trav1 = start1;
while (1) {
    if (trav1 -> link == start2)
        break;
    else
        trav1 = trav1 -> link;
}
```

```
trav2 = start2;
while (1) {
    if (trav2 -> link == start2)
        break;
    else
        trav2 = trav2 -> link;
}
```

```
trav1 -> link = start2;   trav2 -> link = start1;
trav1 = trav2 = null;
start = null;
```

(10)

Merging of two sorted linked lists :-

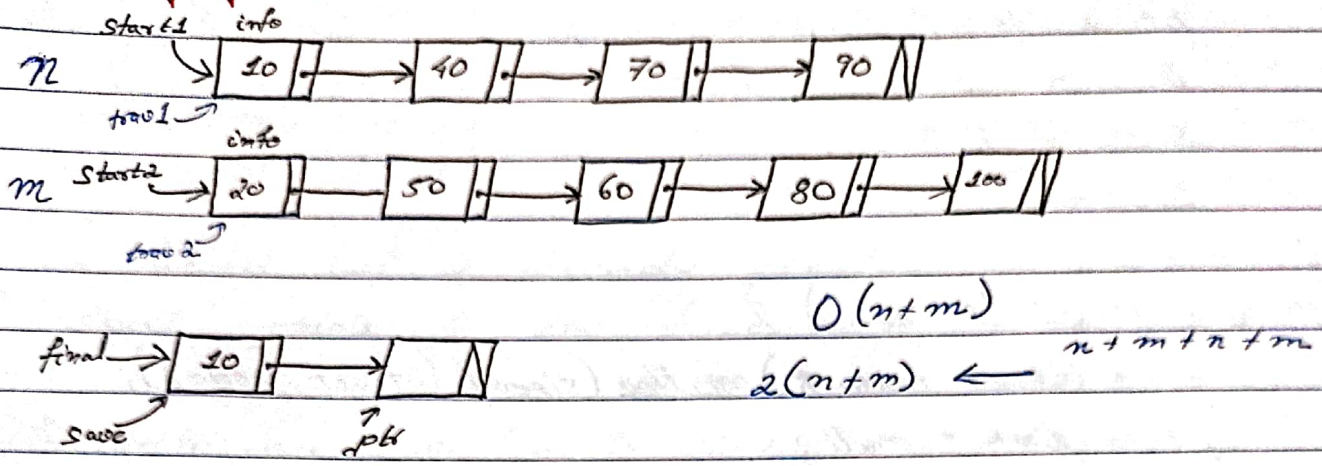



```
final = null;
while ( trav1 != null && trav2 != null ) {
    ptr = (struct node *) malloc (sizeof (struct node));
    if (final == null)
        final = ptr;
    if ( trav1 -> info <= trav1 -> info )
    {
        ptr -> info = trav1 -> info;
        trav1 = trav1 -> link;
    }
    else {
        ptr -> info = trav2 -> info
        trav2 = trav2 -> link;
    }
}
```

Date
17/11/2021

Lecture-22

Merging of two sorted linked lists :-

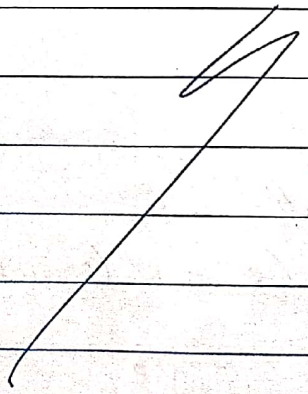


$O(n+m)$

$2(n+m)$ ← $n+m+n+m$

```
trav1 = start1;
trav2 = start2;
ptr = (struct node *) malloc(sizeof(struct node));
ptr → link = null;
if (final == null) → final = ptr;
if (trav1 → info <= trav2 → info) {
        ptr → info = trav1 → info;
        trav1 = trav1 → link;
}
else {
        ptr → info = trav2 → info;
        trav1 = trav1 → link;
}
save = ptr;
while ( trav1 != null && trav2 != null) {
        ptr = (struct node *) malloc(size of (struct node));
        ptr → link = null;
        if (trav1 → info <= trav2 → info) {
                ptr → info = trav1 → info;
                trav1 = trav1 → link;
                save → link = ptr;  save = ptr;
        }
```

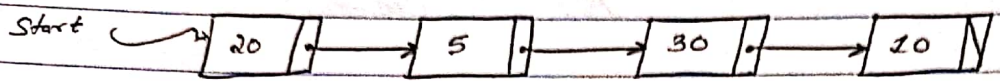
```
else {
    ptr -> info = trav2 -> info;
    trav2 = trav2 -> link;
    save -> link = ptr;
    save = ptr;
}
while ( trav1 != null) {
    ptr = (struct node *) malloc (sizeof (struct node));
    ptr -> link = null;
    ptr -> info = trav1 -> info;
    trav1 = trav1 -> link;
    save -> link = ptr;
    save = ptr;
}
while (trav2 != null) {
    ptr = (struct node *) malloc (sizeof (struct node));
    ptr -> link = null;
    ptr -> info = trav2 -> info;
    trav2 = trav2 -> link;
    save -> link = ptr;
    save = ptr;
}
```



Date
22/11/2021

Lecture-23

How to write an algorithm for linked list:-



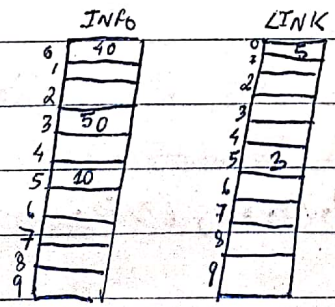
A linked list is already created in memory. The variable 'start' points to the first node of the list.

Q. Write an algorithm to display the list.

Algorithm Display (start)
Begin
1. If (start = null) Then
2. Write ("List is empty")
3. Return
4. End If
5. temp ← start
6. Repeat step 7 through 8 while temp <> null
7. Write ("The content is :", INFO (temp));
8. temp ← LINK (temp);
9. Return

Simulation of a linked list using array:

We take two arrays: INFO and LINK



INFO (temp)
RLINK (temp)
LLINK (temp)

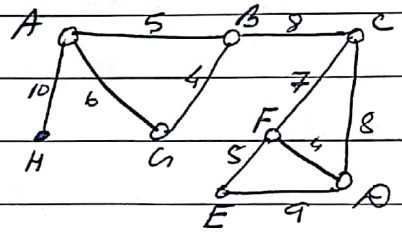
INFO (temp)
NEXT (temp)
PREVIOUS (temp)

Q. Implementation of stack using doubly linked list.
Q. " " queue " " " " .

Non-linear data structures :-

Graph :

- townships
- towns are connected by roads.



Lecture - 24

Date
27/12/2021

Graph :-

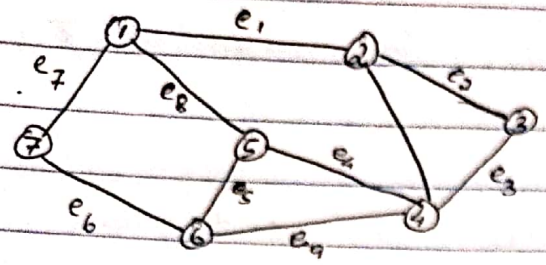
A graph has two sets : V that is the set of vertices (nodes).
E : the set of edges.

There is a mapping from E to V.

e.g. G is a graph

$V(G) = \{1, 2, 3, 4, 5, 6, 7\}$

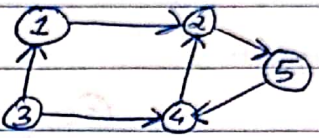
$E(G) = \{(1,2), (2,3), (3,4), (2,4), (4,5), (5,6), (7,5), (1,5)\}$



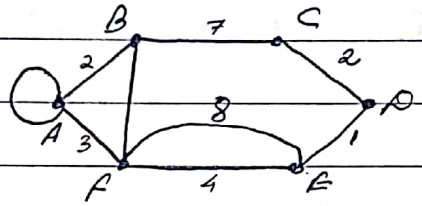
(1,2)
(2,1)

Undirected :- $V_1, V_2 \in V(a)$, then (V_1, V_2) and (V_2, V_1) same.
Directed :- each edge has direction, so (V_1, V_2) and (V_2, V_1) are not same.

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



#Weighted Graph :- each edge carries a weight.



Self Loop : an edge whose starting and end vertices are same.

Parallel Edges : two or more edges between the same two vertices.

Adjacent Vertices : two vertices are adjacent if they are connected by an edge.

Incidence : there is an edge (V_1, V_2) in undirected graph this edge is incident on V_1 and V_2.

Degree of a vertex :- the number of edges incident to that vertex (undirected).

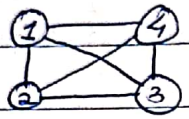
In case of directed graph : indegree and outdegree.

Simple graph : graph without self-loop or parallel edges.

Multigraph : graph with self-loop or parallel edges or both.

Complete Graph :

$\frac{n(n-1)}{2}$ edges



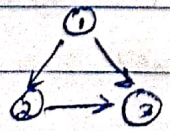
Regular Graph : each vertex is adjacent to the same no. of vertices.

Connected graph : for any two vertices V_1 and V_2, there is a path from V_1 to V_2 or from V_2 to V_1.

path : a sequence of edges.

For a directed graph : if there is a path for each pair of a distinct vertices, it is strongly connected.

Weakly Connected : for any two vertices u and v, there is a path from u to v from u to u.



1,1 ✓
1,3 ✓
2,3 ✓

Cycle : a path in a graph where the starting and end vertices are same.

Cyclic graph: a graph that has cycles

Acyclic graph: a graph that does not have any cycle.

Maximum no. of edges in a graph:

$n(n-1)/2$

$n(n-1)$

//

Lecture-25

Date 29/12/2021

Q. Check whether F' logically implies $A \to C$. For this, compute $(A)^+$ w.r.t. F',

$(A)^+ = \{A, B, D\}$

The attribute C (the RHS) is not present in (A^+). Therefore, the dependency $A \to C$ is lost. and the decomposition is not dependency.

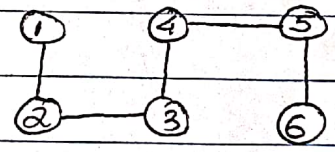
Graph:-

Representation of graph in computer memory:-

1) Set representation:-

$V(G) = \{1, 2, 3, 4, 5, 6\}$

$E(G) = \{(1,2), (2,3), (3,4), (4,5), (5,6)\}$



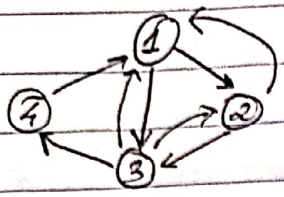
- efficient in terms of memory utilization

2) Adjacency matrix:-

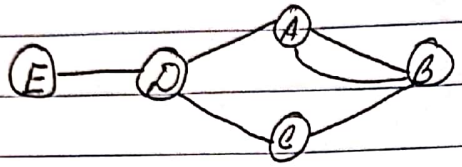
- represent the graph as a matrix

$a_{ij} = 1$, then there is an edge from v_i to v_j

$= 0$ otherwise.



	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	1	0	0	0



n x n space

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

3) Incidence Matrix :-

$a_{ij} = \begin{cases} 1, & \text{if the } j\text{th edge is incident on the } i\text{th vertex } v_i \\ 0, & \text{otherwise} \end{cases}$

e.g.

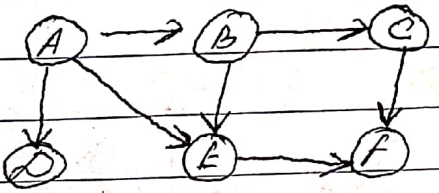
4) Linked representation :-

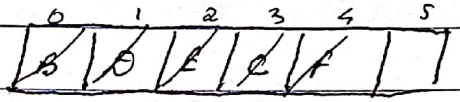
Graph Traversal :- Visiting each vertex of the graph in some order.

Breadth First Search (BFS) :-

- BFS using queue

Start the traversal at node A





A B D E C F

Lecture – 26

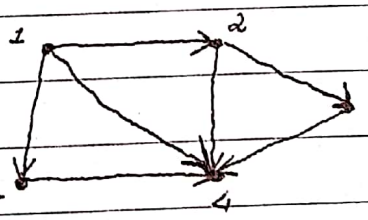
Date 01/22/2021

Graph Traversal :

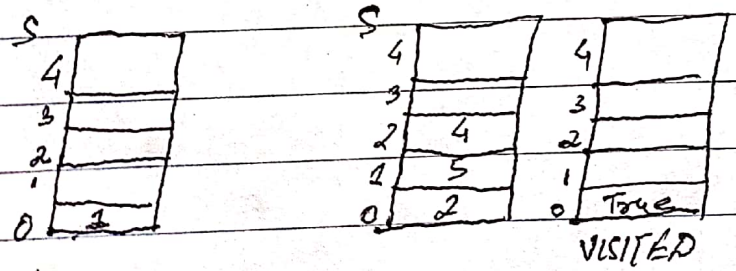
① Breadth first search : Using queue

② Depth first search : Using stack.

e.g.



Let the BFS begin at node 1



Top = 0

(i) (ii) (iii)

(50)

S

4	
3	
2	
1	
0	3

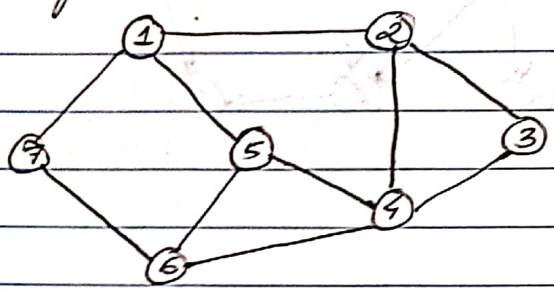
Top = -1

4	True
3	True
2	True
1	True
0	True

Visited

Visit: 1
Visit: 4
Visit 5
Visit: 2
Visit: 3

e.g.



BFS :-

START 1
while (queue not empty) {
    Delete front element & traverse;
    Insert all adjacent nodes into queue if not done already;
}