Unit II
Linked List

A linked list is a very flexible, dynamic data structure in which elements (called nodes) form a sequential list. In a linked list, each node is allocated space as it is added to the list. Every node in the list points to the next node in the list. Therefore, in a linked list, every node contains data of the node and a pointer to next node. The last node in the list contains a NULL pointer to indicate that it is the end or tail of the list. The total number of nodes that may be added to a list is limited only by the amount of memory available.

Applications of linked list in computer science –

1. Implementation of stacks and queues
2. Implementation of graphs : Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices.
3. Dynamic memory allocation : We use linked list of free blocks.
4. Maintaining directory of names
5. Performing arithmetic operations on long integers
6. Manipulation of polynomials by storing constants in the node of linked list
7. representing sparse matrices

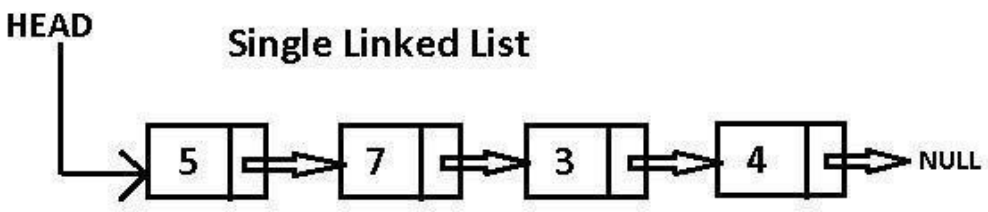
Applications of linked list in real world-

1. Image viewer – Previous and next images are linked, hence can be accessed by next and previous button.
2. Previous and next page in web browser – We can access previous and next url searched in web browser by pressing back and next button since, they are linked as linked list.
3. Music Player – Songs in music player are linked to previous and next song. you can play songs either from starting or ending of the list.

Types of Linked Lists –

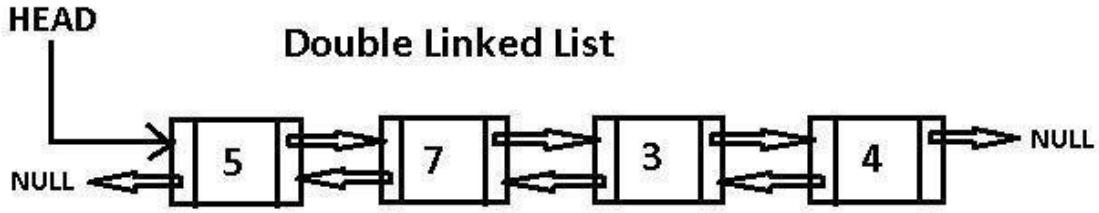
SINGLE LINKED LIST

A single linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node. A singly linked list allows traversal of data only in one way. So these are unidirectional.



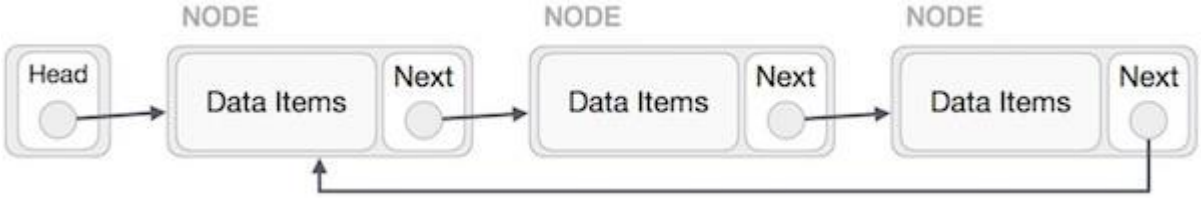
DOUBLE LINKED LIST

A double linked list is similar to single linked list but has two pointers in the node, where the next pointer points to next node and the previous pointer points to previous node. A double linked list as it has next and previous pointers , we can traverse in both directions, so these are bi-directional.



CIRCULAR LINKED LIST :

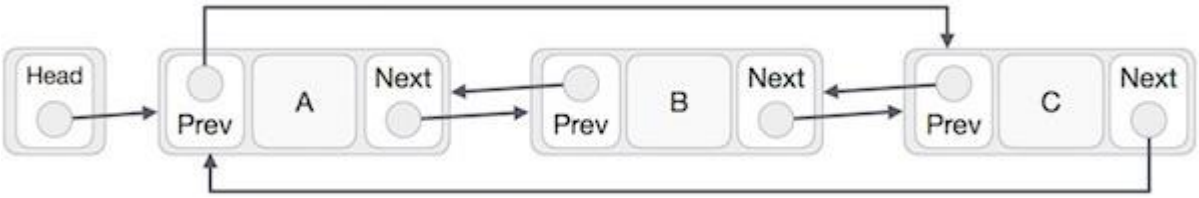
Singly Linked List as Circular
In singly linked list, the next pointer of the last node points to the first node.



The next pointer of the last node points to the first node.

Doubly Linked List as Circular

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



As per the above illustration, following are the important points to be considered.

- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.

- The first link's previous points to the last of the list in case of doubly linked list.

Q. Explain Linked list applications and types of linked lists.

There can be questions on only applications of LL or types of LL for 5 marks
There can also be 5 marks questions like explain singly circular linked list or doubly circular linked list or double linked list etc.

Implementation of Single Linked List ADT:

Following are the important operations supported by a linked list. The ADT operations are:

- **create** – Creates a linked list.
- **insert** – Inserts an element at a specified location in the list.
- **delete** – Deletes a specified element from the list.
- **display** – Displays all elements of the list.

```
create_node(data)
        Begin
        create a new node
        node -> data := data
        node -> next := NULL

        if the list is empty, then
        head := node
        else
        temp := head
        while next of temp is not NULL, do
        temp := next of temp
        done
        next of temp := node
        end if
        End

insert(data, position)
        Begin
        create a new node
        node -> data := data
        node -> next := NULL
```

```
        if position = 1 then
        node -> next := head
        head := node

        else
        temp := head
        count := 1

        while count < position - 1 AND temp is not NULL do
        temp := temp -> next
        count := count + 1
        done

        if temp is NULL then
        print "Invalid Position"
        else
        node -> next := temp -> next
        temp -> next := node
        end if
        end if
        End

traverse()
        Begin
        if head = NULL then
        print "List is empty"
        else
        temp := head
        while temp is not NULL do
        print temp -> data
        temp := temp -> next
        done
        end if
        End

search(key)
        Begin
        if head = NULL then
        print "List is empty"
        return
        end if
```

```
temp := head
position := 1

while temp is not NULL do
if temp -> data = key then
print "Element found at position", position
return
end if

temp := temp -> next
position := position + 1
done

print "Element not found"
End
```

Q. Explain Linked list ADT

There can be questions for 5 marks lik e:

Q. Write about operations on LL. Or

Explain insert operation or deletion operation etc.
