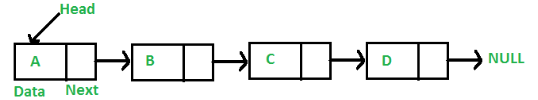
# Linked List:

It is similar to array difference is the storing of memory of its next element and easily variable size. Each element will act as separate object.

**Train:**

Engine -> Compartment 1-> Compartment2 -> Compartment3-> Guard room



## How is the linked list build?

Linear data structure where data elements is a separate object. Each has element (node/data) and reference. With option of adding/removal of size.

## Advantages and Disadvantages:

### Advantage:

Save space and easy to maintain.

No need to allocate the size

### Disadvantage:

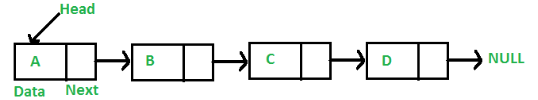
Can’t go backwards or access easily.

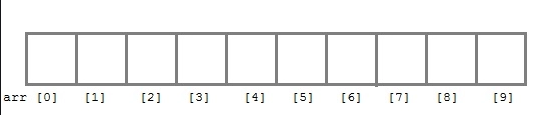
## Application:

Application of computer and memory storage

# Difference between Array and Linked List

* Every element is a separate object in Linked List. But array is an object as a whole.
* Never delete a space.
* Variable size.
* Random access



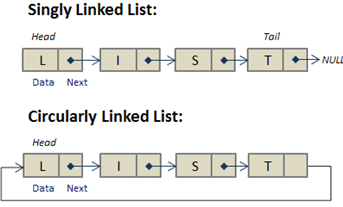


## Components of Linked List:

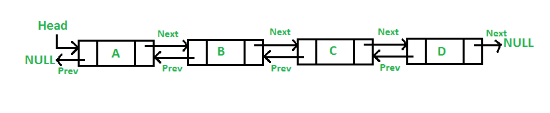
* Node
* Head
* Tail

# Types of linked list

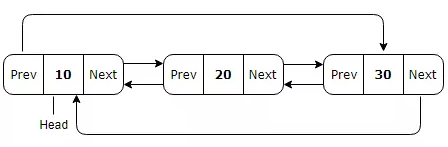
* Single
* Double
* Circular single
* Circular double.



**Double Linked List:**



**Double Circular:**

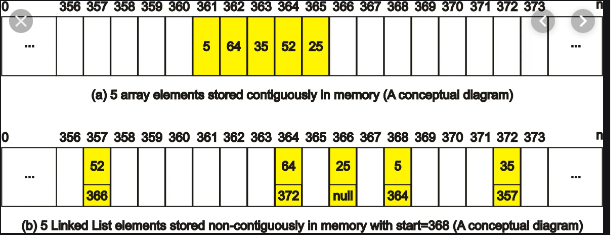


## Why so many linked list

Singular – Board game- 1,2,3,4

Double – Music Player – 1…. 100

# How memory is stored in Linked List:



# Creation of Linked List:

Creation of Linked List (NodeValue):

Create a head, tail and initialize the values as Null

Create a blank node

Node.value = NodeValue

Node.next = null

Head = tail = node

**Representation:**

|  |  |  |
| --- | --- | --- |
| Head | Node | Tail |

Time Complexity: O(1)

Space complexity - O(1)

# Insertion of Linked List:

Insertion can be made in 3 cases:

Insert a new node at start of the linked List

Insert at the end of the Linked list

Insert in the specific location of Linked list.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Head | Node1 – Node2 | Node2 – Node3 | Node3 | Tail |

Insertion Algorithm:

InsertLL(head,node,location)

Create a blank node

Node value = node

If linkedlist does not exist

Return error

Else if (Location =0)

Node.next= head

Head = node

Else if (Location =last)

Node.next= null

Last .next= node

Last = node

Else:

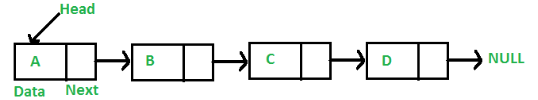
Loop from 0 to node -1

Node.next= tmpnode.next

Tmpnode.next = node

**Time complexity – O (n); Space Complexity is O (1)**

## Traversal



**Algorithm:**

TraverseLL(head)

If head is null

Return error

Else:

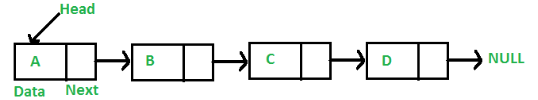
Loop from head to tail

Print current node.

Time Complexity: O(n)

Space Complexity: O(1)

# Search of Linked List



**Algorithm:**

SearchLL(node)

If head is null

Return error

Else:

Loop from head to tail

If currentnode = node

Print current node

Else return NA.

Time Complexity: O(n)

Space Complexity: O(1)

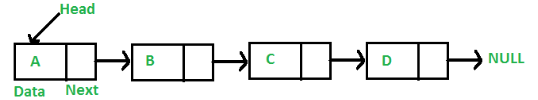
## Deletion of Linked list:

Deletion can be made in 3 cases:

Deletion a node at start of the linked List

Deletion at the end of the Linked list

Deletion in the specific location of Linked list.



|  |  |  |  |
| --- | --- | --- | --- |
| Head – Node2 | Node1 – Node3 | Node3 -Tail | Tail |

DeleteLL(head,node,location)

If LL does not exist:

Return error.

Else if (Location =0)

Head= head.next

If this is the only element then update tail = null

Else if (Location =last)

If this is the only element then update tail = head = null

Loop till 2nd last element (tmp)

Tail = tmp

Tmp.next = null

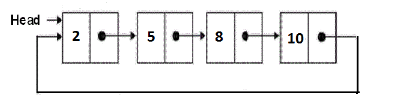
Else

Loop : tmpnode= start to location-1

Tmpnode.next = Tmpnode.next.next

**Time complexity – O (n); Space Complexity is O (1)**

# Creation of circular linked list:



Head and Node.

Creation:

Head Node tail

**Logic:**

Creation of Linked List (NodeValue):

Create a blank node

Node.value = NodeValue # (Head = tail)

Node.next = node

Head = tail = node

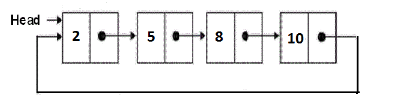
# Insert:

Insertion can be made in 3 cases:

Insert a new node at start of the linked List

Insert at the end of the Linked list

Insert in the specific location of Linked list.



Tail

Node

Head

Tail

Node

Node

Head

Tail

Node

Node

Node

Head

## Insertion Algorithm:

InsertCLL(head,node,location)

Create a blank node

Node value = node

If linkedlist does not exist

Return error

Else if (Location =0)

Node.next= head

Tail.next = node

Else if (Location =last)

Node.next= head

tail.next= node

Tail = node

Else:

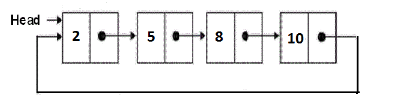
Loop from 0 to node -1

Node.next= tmpnode.next

Tmpnode.next = node

**Time complexity – O (n); Space Complexity is O (1)**

# Traversal of CLL



**Algorithm:**

TraverseCLL(head)

If head is null

Return error

Else:

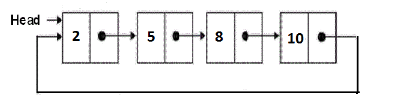
Loop from head to tail

Print current node.

Time Complexity: O(n)

Space Complexity: O(1)

# Search of CLL



**Algorithm:**

SearchCLL(node)

If head is null

Return error

Else:

Loop from head to tail

If currentnode = node

Print current node

Else return NA.

Time Complexity: O(n)

Space Complexity: O(1)

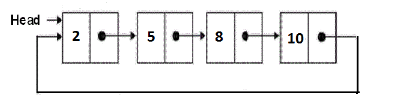
# Deletion of CLL

Deletion can be made in 3 cases:

Deletion a node at start of the linked List

Deletion at the end of the Linked list

Deletion in the specific location of Linked list.



DeleteCLL(head,node,location)

If LL does not exist:

Return error.

Else if (Location =0)

Head= head.next

If this is the only element then update tail = null

Else if (Location =last)

If this is the only element then update tail = head = null

Loop till 2nd last element (tmp)

Tail = tmp

Tmp.next = null

Else

Loop : tmpnode= start to location-1

Tmpnode.next = Tmpnode.next.next

**Time complexity – O (n); Space Complexity is O (1)**

Time complexity

Creation 🡪 O(1)

Insertion 🡪 O(1) Insert at specific position 🡪 O(n)

Searching 🡪 O(n)

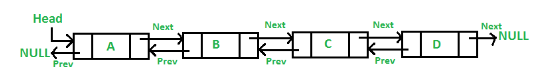
Deletion 🡪 O(1)

Traversing 🡪 O(n)

# Creation of Doubly Linked list:

It is similar to Singly Linked list difference is the each node storing two reference. One at the front and one at the back.

Its variable size. Each element will act as separate object.



|  |  |  |
| --- | --- | --- |
| Head – Node 1 | Head – Node 1 - Tail | Tail - Null |

**Logic**:

CreateDLL (Value)

Create a blank node

Node.value = Value

Head = Tail = node

Node.next = null

Node.prev = null

**Time Complexity – O(1) Space Complexity – O(1)**

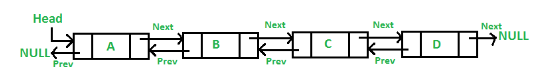
# Insertion of elements in DLL

Insertion can be made in 3 cases:

Insert a new node at start of the linked List

Insert at the end of the Linked list

Insert in the specific location of Linked list.



**Algorithm**:

InsertLL(head,node,location)

Create a blank node

Node value = node

If linkedlist does not exist

Return error

Else if (Location =0)

Node.next= head

Node.prev = null

Head.prev= node

Head = node

Else if (Location =last)

Node.next= null

node.prev= tail

Tail.next = node

Tail = node

Else:

Loop from 0 to location -1

Node.next= tmpnode.next

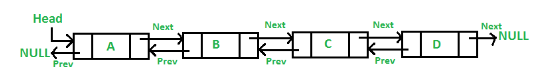
Node.prev = tmpnode

Tmpnode.next = node

Tmpnode.next.prev = node

**Time complexity – O (n); Space Complexity is O (1)**

# Traversal of DLL



**Algorithm:**

TraverseDLL(head)

If head is null

Return error

Else:

Loop from head to tail

Print current node.

Time Complexity: O(n)

Space Complexity: O(1)

TraverseDLL(tail)

If tail is null

Return error

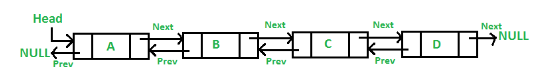
Else:

Loop from tail to head

Print current node.

**Time Complexity: O (n) Space Complexity: O(1)**

# Search of DLL



SearchDLL(node)

If head is null

Return error

Else:

Loop from head to tail

If currentnode = node

Print current node

Else return NA.

Time Complexity: O(n)

Space Complexity: O(1)

ReverseSearchDLL(node)

If head is null

Return error

Else:

Loop from tail to head

If currentnode = node

Print current node

Else return NA.

Time Complexity: O(n)

Space Complexity: O(1)

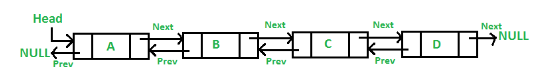
# Deletion of DLL

Deletion can be made in 3 cases:

Deletion a node at start of the linked List

Deletion at the end of the Linked list

Deletion in the specific location of Linked list.



DeleteDLL(head,node,location)

If LL does not exist:

Return error.

Else if (Location =0)

Head= head.next

Head.prev =null

If this is the only element then update head =tail = null

Else if (Location =last)

If this is the only element then update tail = head = null

Tail = tail.prev; tail.next = null

Tmp.next = null

Else

Loop : tmpnode= start to location-1 # Prev node from the node to be deleted

Tmpnode.next = Tmpnode.next.next; Tmpnode.next.prev = tmpnode

**Time complexity – O (n); Space Complexity is O (1)**

## Deletion of DLL:

DeleteDDLL(head, tail)

Loop (tmp: head to tail): tmp.prev = null

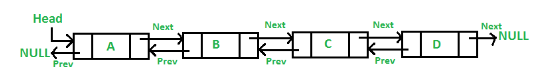
Head= tail = null

# Creation of DCLL

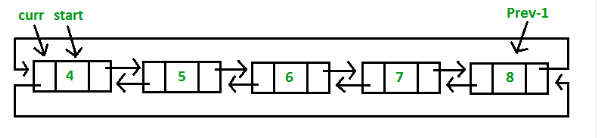
It is similar to Double Linked list with only difference being the last node is connected to the first and vice versa.

Its variable size. Each element will act as separate object.

**Double Linked List**



**Doubly Circular Linked List:**



|  |  |  |
| --- | --- | --- |
| Tail -**Head** – Node 1 | Node1 – **Node 1** – Node1 | Node1 -**Tail** - Head |

**Logic**:

CreateDLL (Value)

Create a blank node

Node.value = Value

Head = Tail = node

Node.next = Node.prev = node

**Time Complexity – O(1) Space Complexity – O(1)**

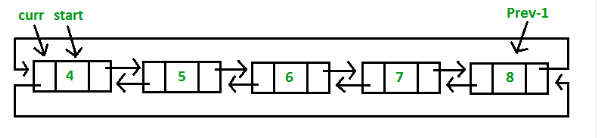
# Insertion of elements in DCLL

Insertion can be made in 3 cases:

Insert a new node at start of the linked List

Insert at the end of the Linked list

Insert in the specific location of Linked list.



**Algorithm**:

InsertLL(head,node,location)

Create a blank node

Node value = node

If linkedlist does not exist

Return error

Else if (Location =0)

Node.next= head

Node.prev = tail

Head.prev= node

Head = node

Tail.next = node

Else if (Location =last)

Node.next= head; node.prev= tail

Tail.next = node

Tail = node;

Head.prev = code

Else:

Loop from 0 to node -1

Node.next= tmpnode.next

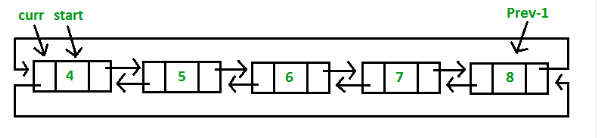
Node.prev = tmpnode

Tmpnode.next = node

node.next.prev = node

**Time complexity – O (n); Space Complexity is O (1)**

# Traversal of DCLL



**Algorithm:**

TraverseDCLL(head)

If head is null

Return error

Else:

Loop from head to tail

Print current node.

**Time Complexity: O(n) Space Complexity: O(1)**

TraverseDLL(tail)

If tail is null

Return error

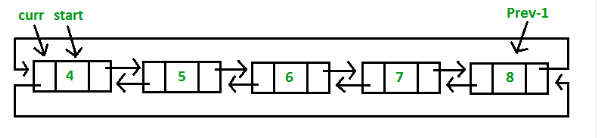
Else:

Loop from tail to head

Print current node.

**Time Complexity: O(n) Space Complexity: O(1)**

# Search of DCLL



SearchDCLL(node)

If head is null

Return error

Else:

Loop from head to tail

If currentnode = node

Print current node

Else return NA.

**Time Complexity: O(n) Space Complexity: O(1)**

ReverseSearchDLL(node)

If head is null

Return error

Else:

Loop from tail to head

If currentnode = node

Print current node

Else return NA.

**Time Complexity: O(n) Space Complexity: O(1)**