Data Structure & Algorithm Design

CT-159

Prepared by
Muhammad Kamran

Linked List

- A **linked list** is a linear data structure that consists of a series of nodes connected by pointers (in C or C++) or references (in Java, Python and JavaScript).
- Each node contains **data** and a **pointer**/**reference** to the next node in the list.
- Unlike **arrays, linked lists** allow for efficient **insertion** or **removal** of elements from any position in the list, as the nodes are not stored contiguously in memory.

Linked List vs Arrays

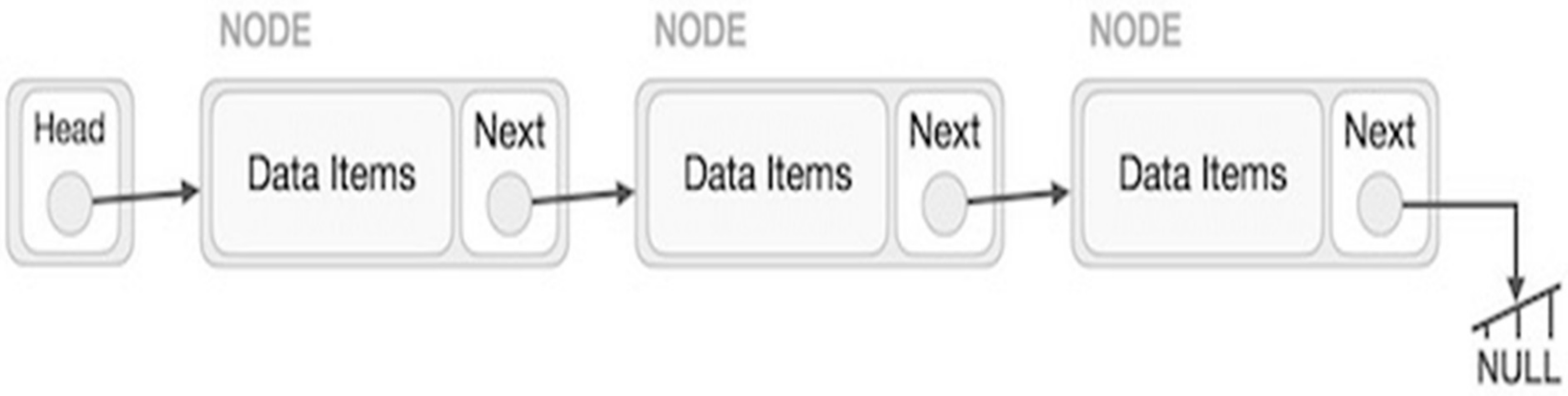
- **Data Structure:** Non-contiguous
- **Memory Allocation:** Typically allocated one by one to individual elements
- **Insertion/Deletion:** Efficient
- **Access:** Sequential

- **Data Structure:** Contiguous
- **Memory Allocation:** Typically allocated to the whole array
- **Insertion/Deletion:** Inefficient
- **Access:** Random

Linked List

- A linked list is a sequence of data structures, which are connected together via links.
- Linked List is a sequence of links which contains items.
- Each link contains a connection to another link.
- Linked list is the second most-used data structure after array.

Linked List Representation



Important Points

- Linked List contains a link element called Head.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

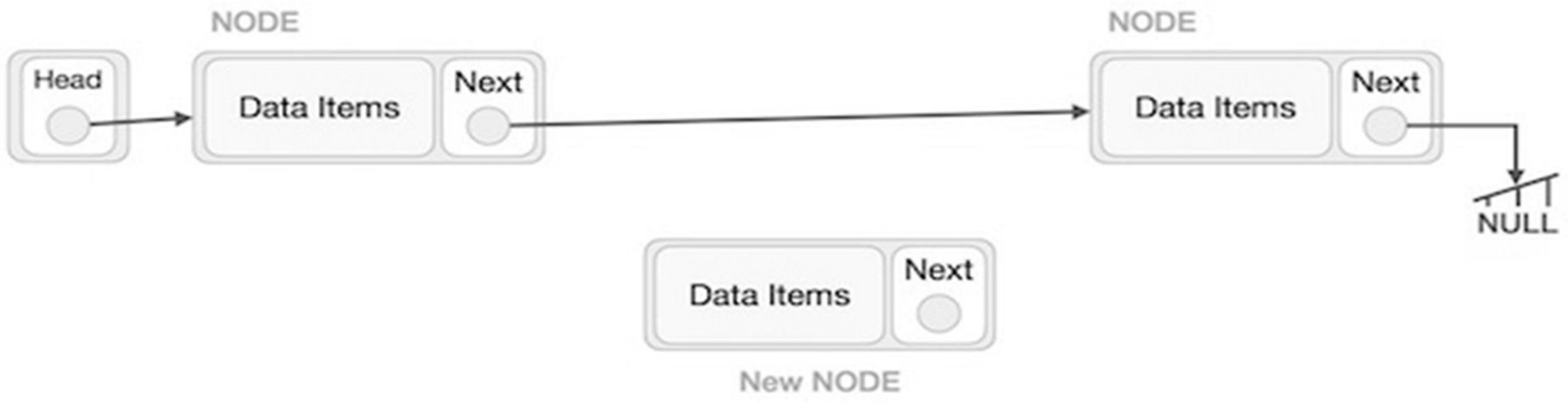
Types of Linked List

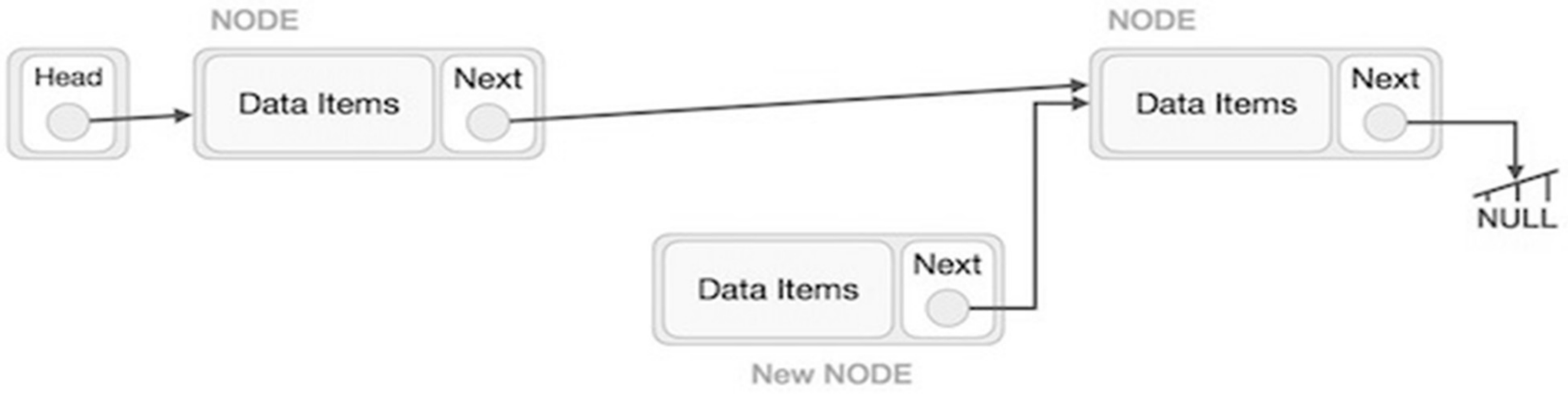
- **Simple Linked List** – Item navigation is forward only.
- **Doubly Linked List** – Items can be navigated forward and backward.
- **Circular Linked List** – Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

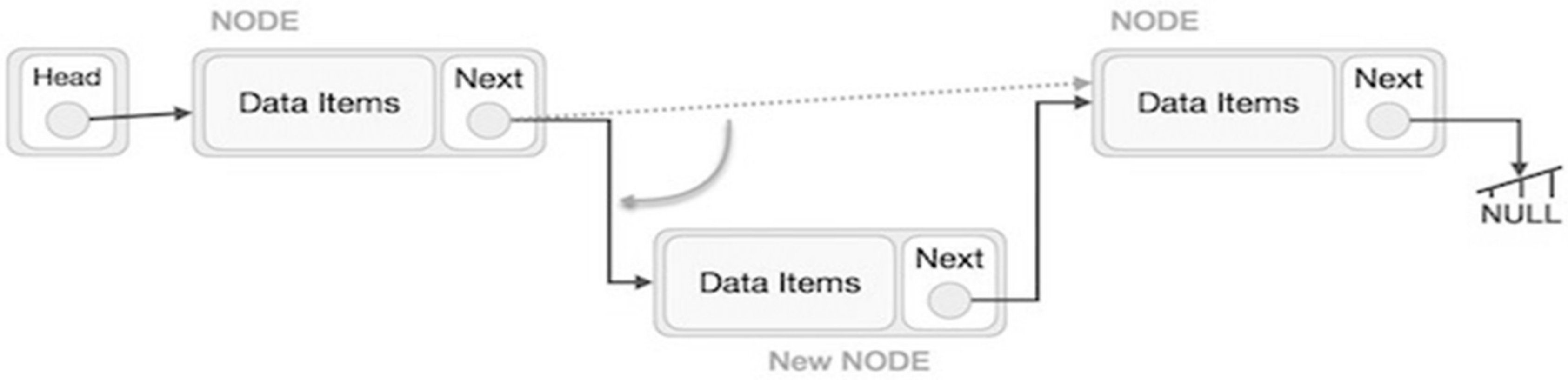
- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.
- **Delete** − Deletes an element using the given key.

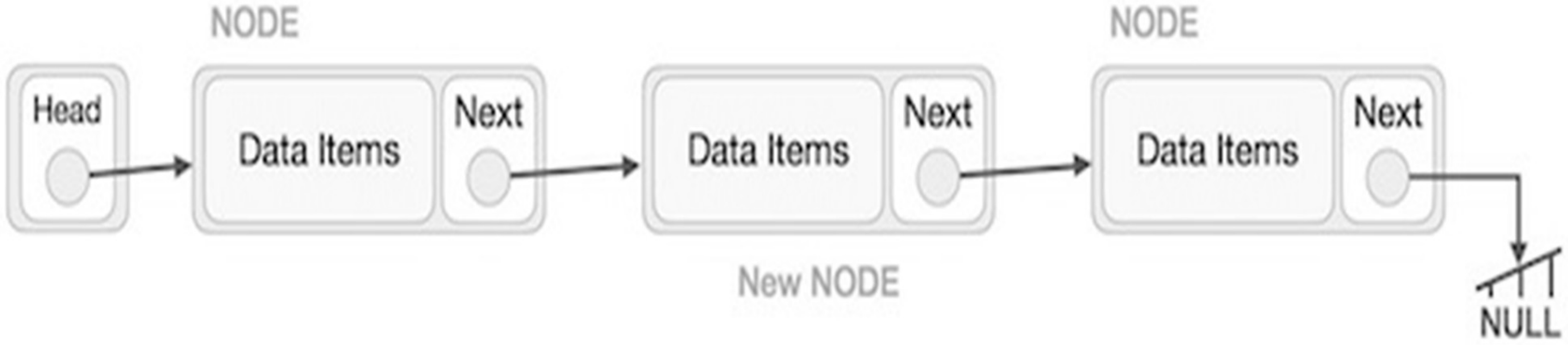
Insertion


NODE
NODE
Head
Data Items
Next
Data Items
Next
NULL
Data Items
Next
New NODE

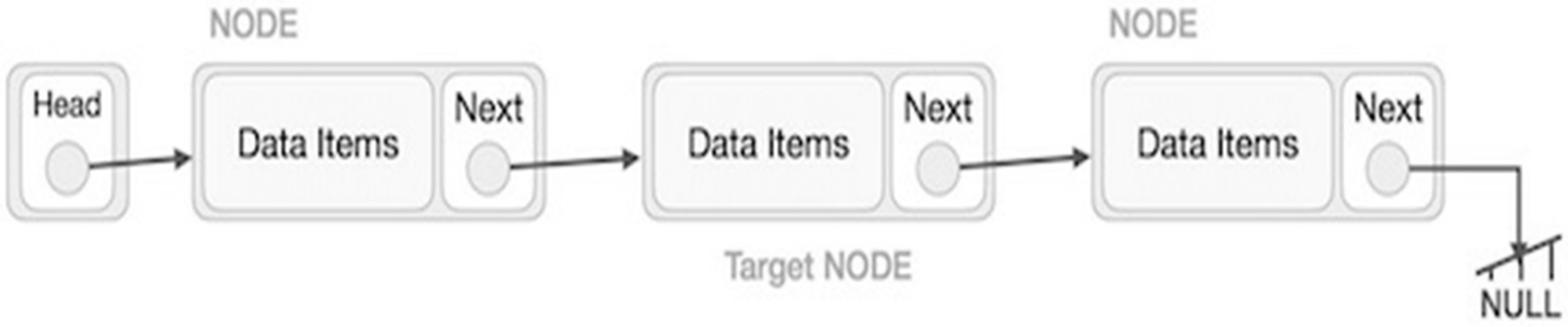

NODE
NODE
Head
Data Items
Next
Data Items
Next
NULL
Data Items
Next
New NODE

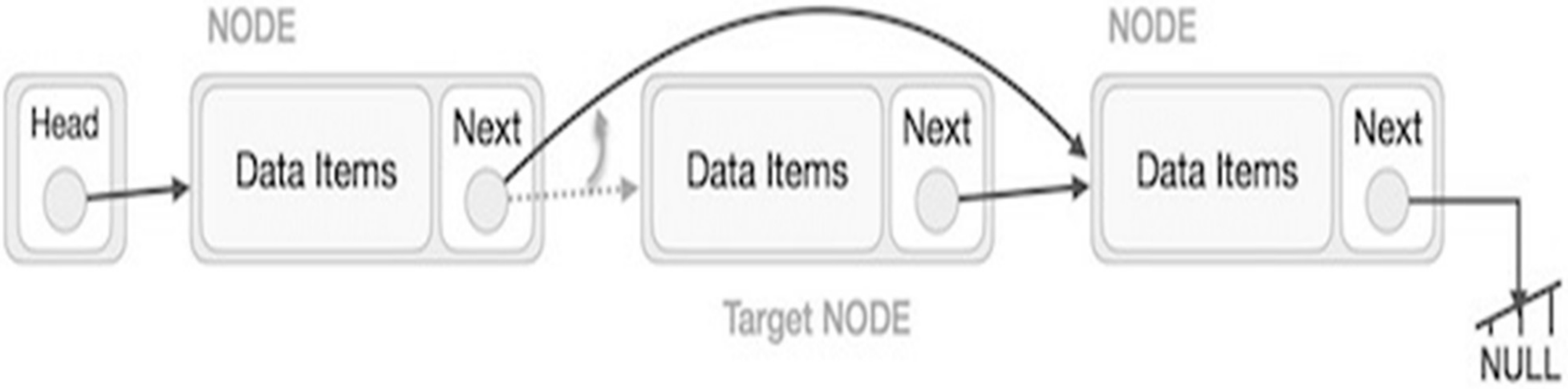
Insertion


NODE
Head
Data Items
Next
NODE
Data Items
Next
NULL
Data Items
Next
New NODE

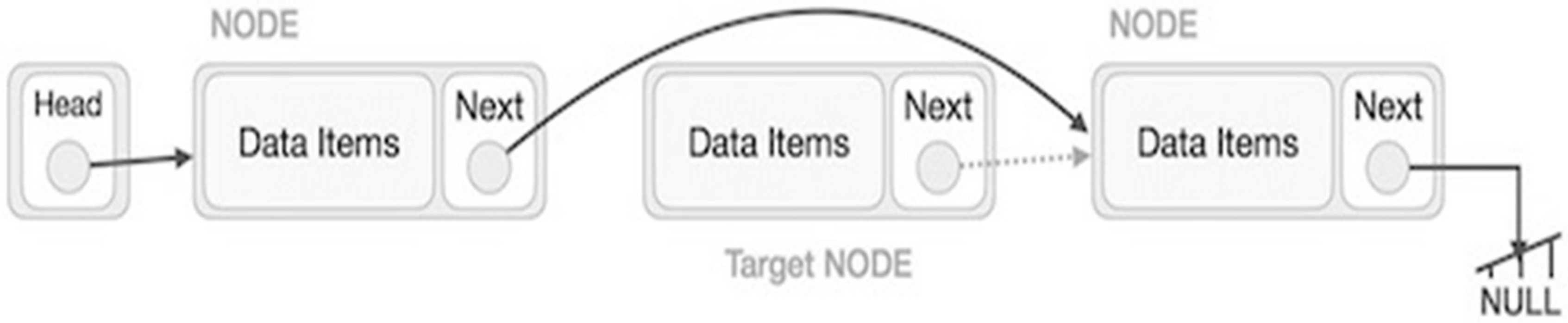

NODE
NODE
Head
Data Items
Next
Data Items
Next
Data Items
Next
New NODE
NULL

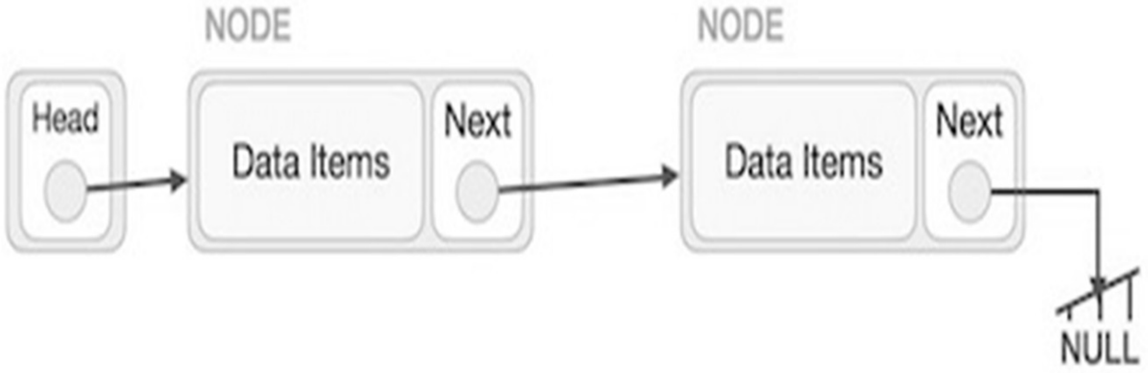
Deletion


NODE
NODE
Head
Data Items
Next
Data Items
Next
Data Items
Next
Target NODE
NULL


NODE
NODE
Head
Data Items
Next
Data Items
Next
Data Items
Next
Target NODE
NULL

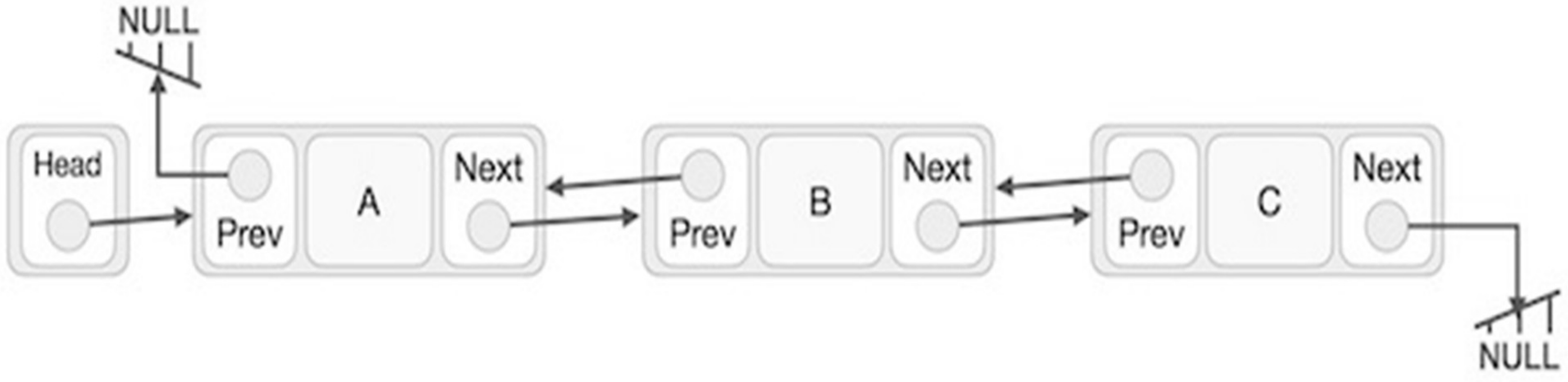
Deletion


NODE
NODE
Head
Data Items
Next
Data Items
Next
Data Items
Next
Target NODE
NULL


NODE
NODE
Head
Data Items
Next
Data Items
Next
NULL

Doubly Linked List

- Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List.



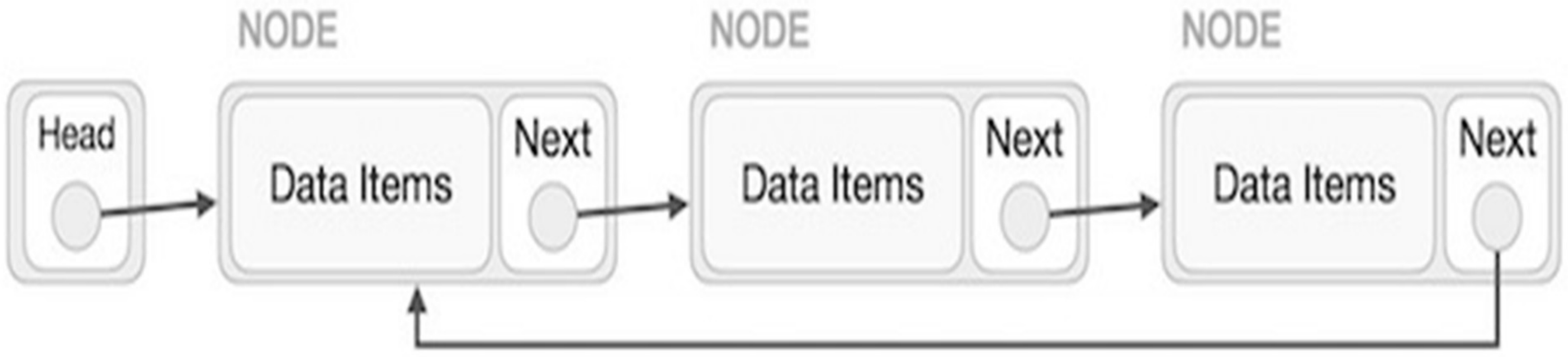
Important Points

- Doubly Linked List contains a link element called first and last.
- Each link carries a data field(s) and two link fields called next and prev.
- Each link is linked with its next link using its next link.
- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.

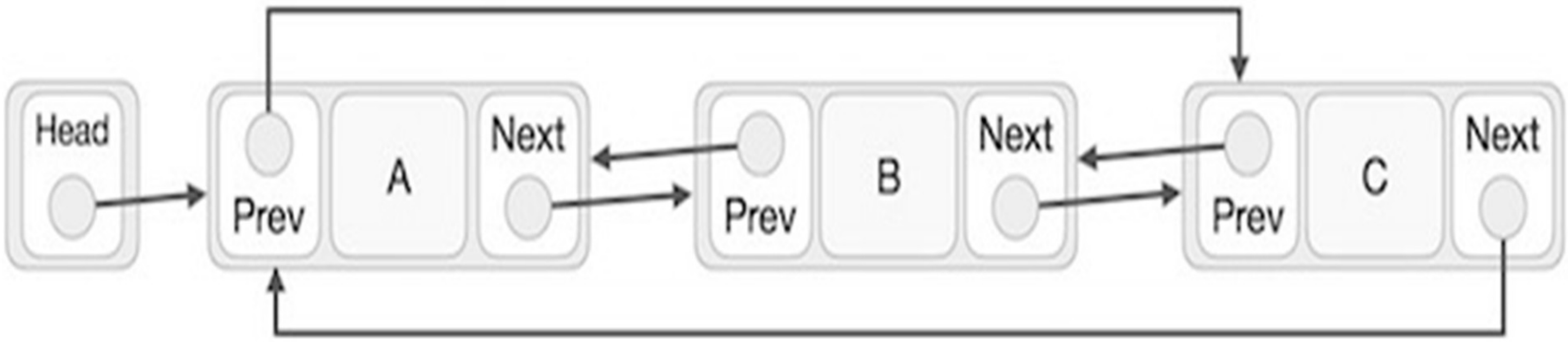
Circular Linked List

- Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

Singly Linked List as Circular



Doubly Linked List as Circular

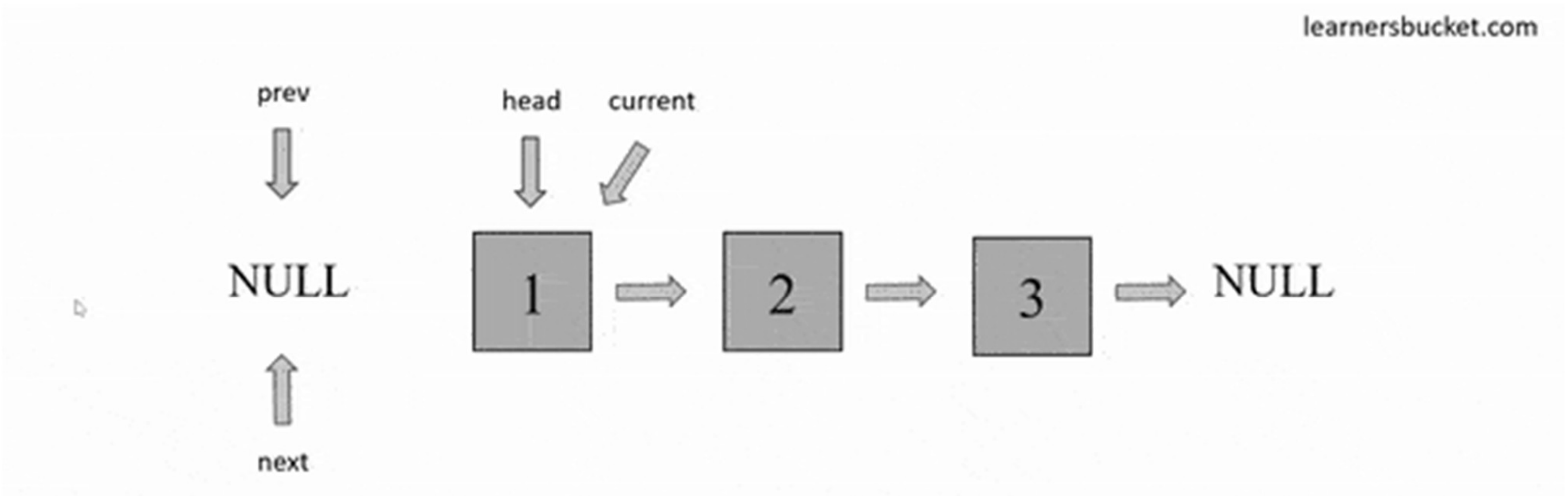


How to reverse linked list ???

To Reverse Linked list

- Initialize three pointers prev as NULL, curr as head, and next as NULL.
- Iterate through the linked list. In a loop, do the following:
- Store the next node, next = curr -> next
- Update the next pointer of curr to prev, curr -> next = prev
- Update prev as curr and curr as next, prev = curr and curr = next

To Reverse Linked list



```
while(current !== null){
    next = current.next;
    current.next = prev;
    prev = current;
    current = next;
}
head = prev;
```

Advantages

- Dynamic size
- Efficient Insertion and Deletion
- Memory Efficiency
- Easy to Implement
- Flexibility
- Easy to navigate

Disadvantages

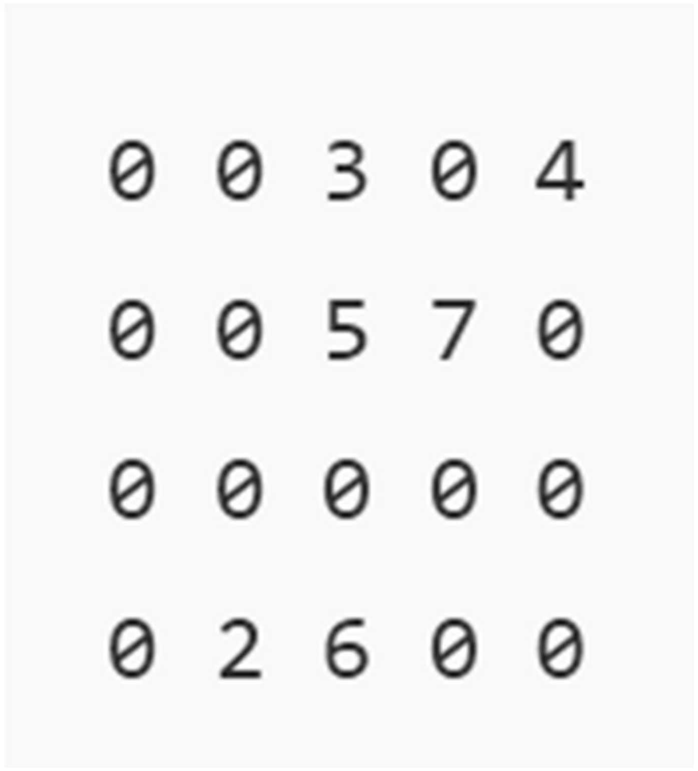
- Slow Access Time
- Pointers
- Higher overhead
- Cache Inefficiency
- Extra memory required

Applications

- Linked Lists are used to implement stacks and queues.
- It is used for the various representations of trees and graphs.
- It is used in dynamic memory allocation(linked list of free blocks).
- It is used for representing sparse matrices.
- It is used for the manipulation of polynomials.
- It is also used for performing arithmetic operations on long integers.
- It is used for finding paths in networks.
- In operating systems, they can be used in Memory management, process scheduling and file system.

Sparse Matrix

- If most of the elements of the matrix have 0 value, then it is called a sparse matrix.



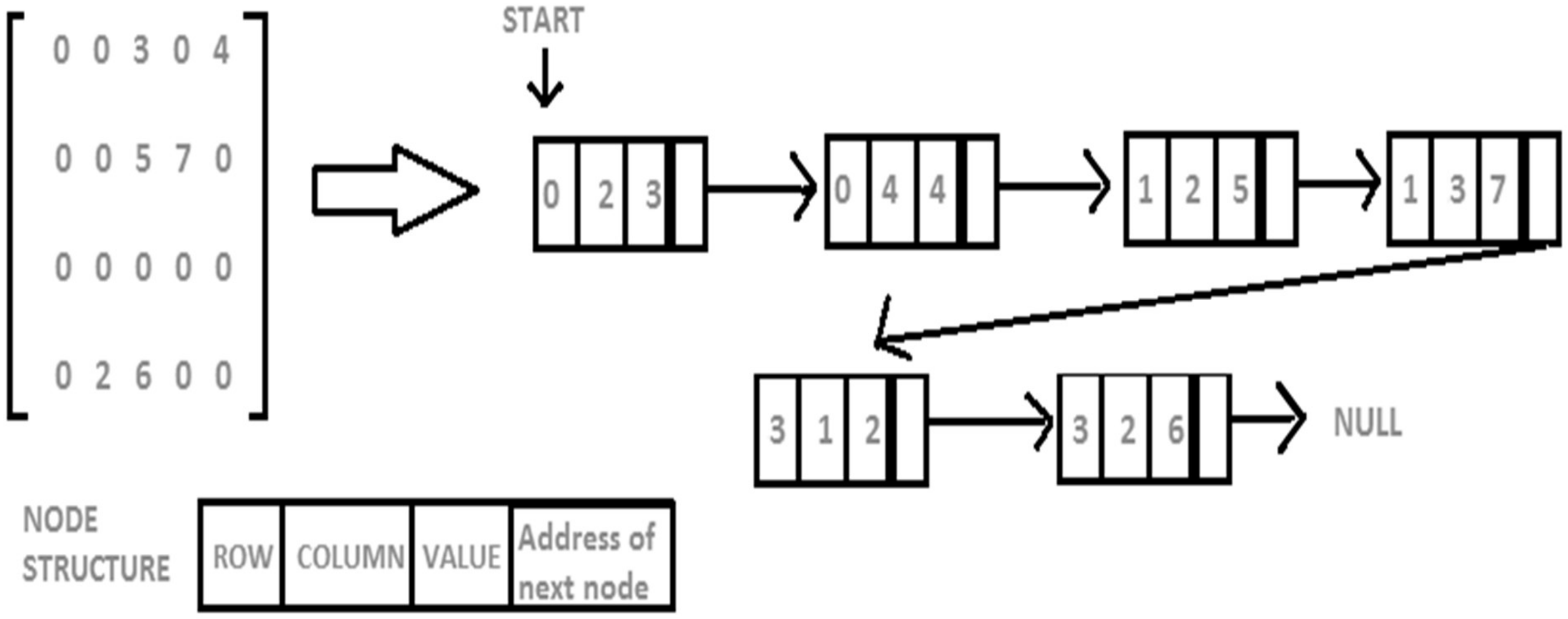
```
0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0
```

Using Arrays

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Using Linked Lists



Homework

1. Given a singly linked list. The task is to remove duplicates (nodes with duplicate values) from the given list (if it exists).

Thank You