Data Structures and Algorithms

Lecture 7: Linked List – Introduction

Linked List - Introduction

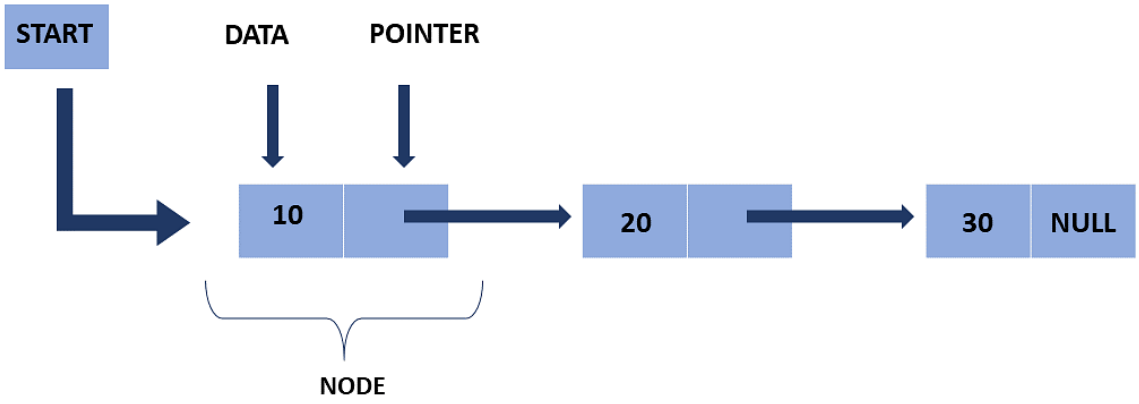
- A linked list is a linear data structure that stores a collection of data elements dynamically.
- Nodes represent those data elements, and links or pointers connect each node.
- Each node consists of two fields, the information stored in a linked list and a pointer that stores the address of its next node.
- The last node contains null in its second field because it will point to no node.
- A linked list can grow and shrink its size, as per the requirement.
- It does not waste memory space.

Linked List - Need

- Dynamic Data structure: The size of memory can be allocated or de-allocated at run time based on the operation insertion or deletion.
- Ease of Insertion/Deletion: The insertion and deletion of elements are simpler than arrays since no elements need to be shifted after insertion and deletion, Just the address needed to be updated.
- Efficient Memory Utilization: As we know Linked List is a dynamic data structure the size increases or decreases as per the requirement so this avoids the wastage of memory.
- Implementation: Various advanced data structures can be implemented using a linked list like a stack, queue, graph, hash maps, etc.

Linked List - Representation

This representation of a linked list depicts that each node consists of two fields. The first field consists of data, and the second field consists of pointers that point to another node.

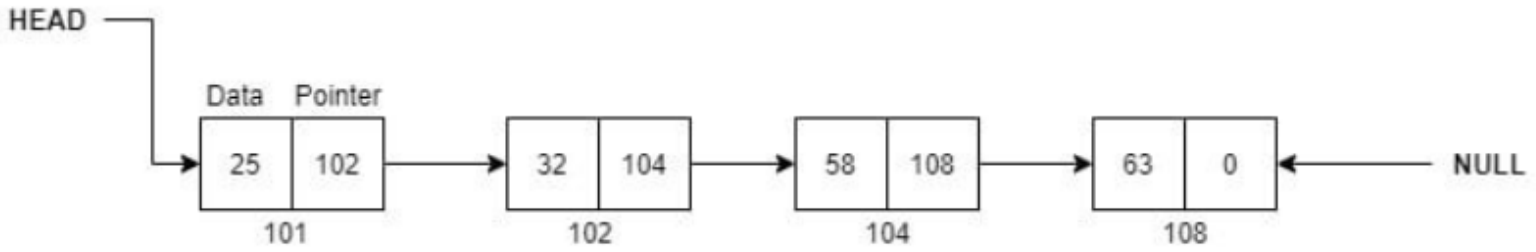


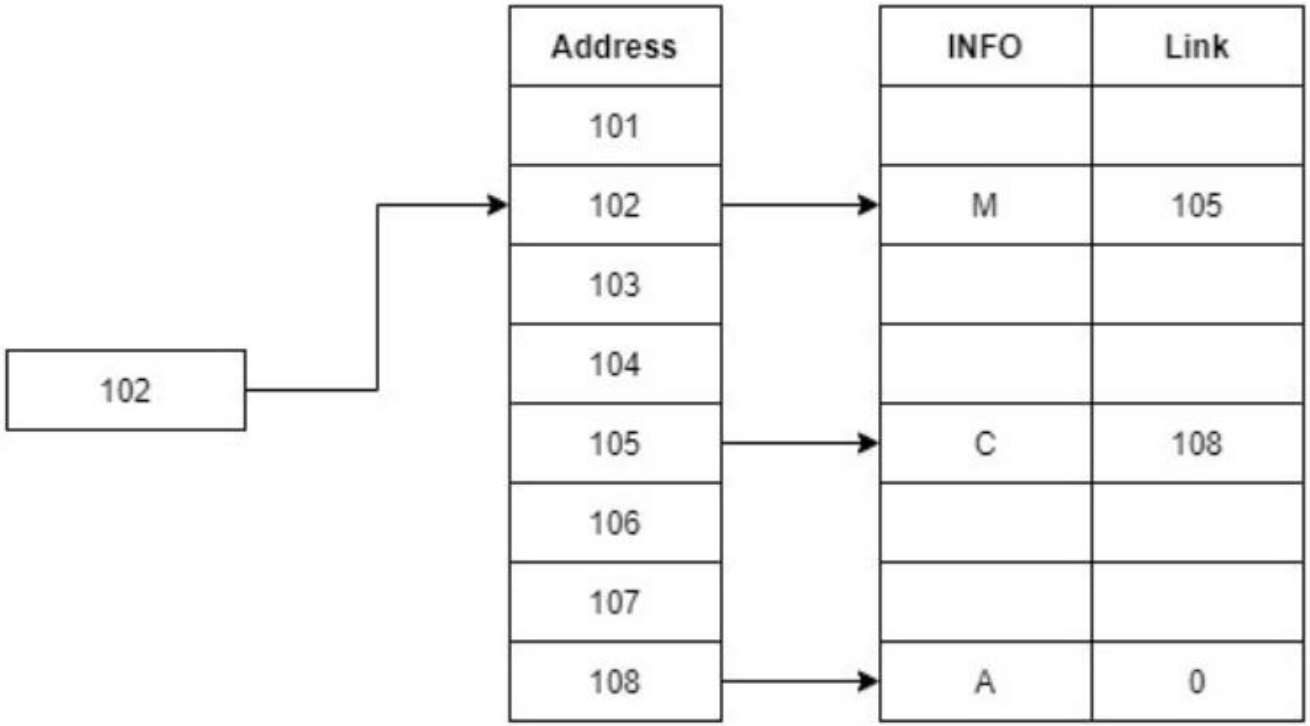
Here, the start pointer stores the address of the first node, and at the end, there is a null pointer that states the end of the Linked List.

Linked List – Memory Representation

- The linked list is stored in different locations in the memory.
- The memory is allocated dynamically for each node.
- Dynamic means that it is allocated when it is needed.
- Due to the dynamical allocation, the user can increase and decrease the size of the linked list whenever required.

Linked List – Memory Representation





Linked List – Creation of Node

```
struct node
{
        int data;
        struct node * next;
};
struct node * n;
n=(struct node*)malloc(sizeof(struct node*));
```

- It is a declaration of a node that consists of the first variable as data and the next as a pointer, which will keep the address of the next node.
- Here you need to use the malloc function to allocate memory for the nodes dynamically.

Linked List – Types

The linked list mainly has three types, they are:

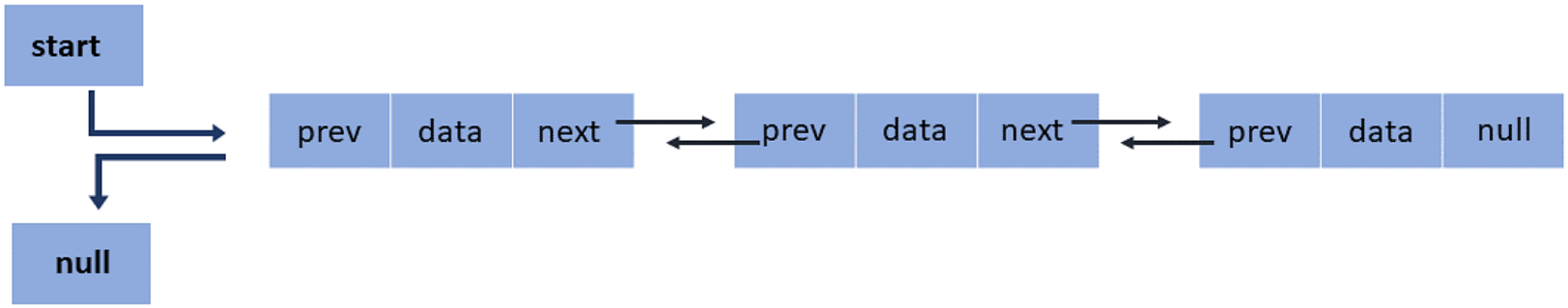
1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Singly Linked List

In a singly linked list, each node contains a reference to the next node in the sequence. Traversing a singly linked list is done in a forward direction.

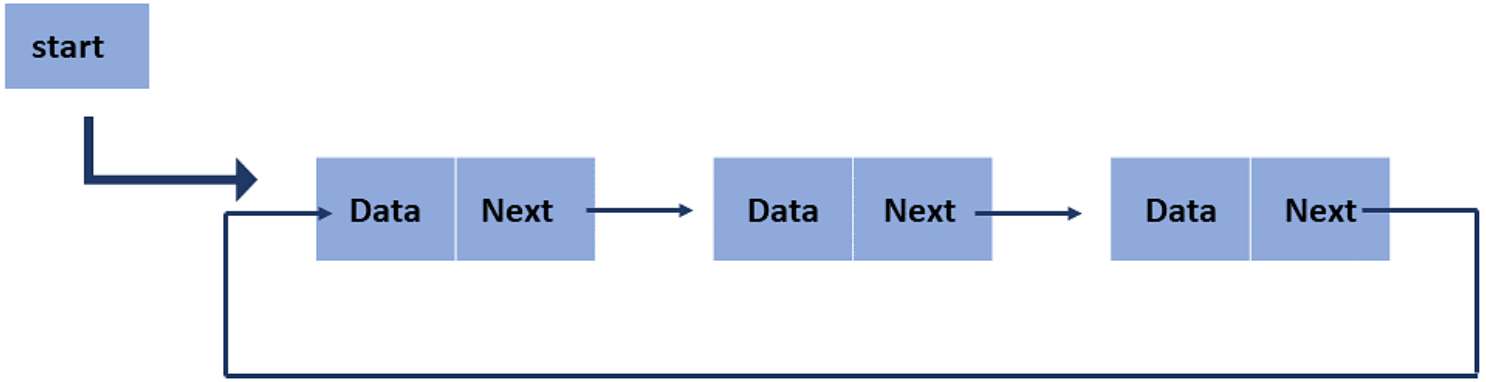
Doubly Linked List

In a doubly linked list, each node contains references to both the next and previous nodes. This allows for traversal in both forward and backward directions, but it requires additional memory for the backward reference.



Circular Linked List

In a circular linked list, the last node points back to the head node, creating a circular structure. It can be either singly or doubly linked.



- Circular link lists can either be singly or doubly-linked lists.
- The next node's next pointer will point to the first node to form a singly linked list.
- The previous pointer of the first node keeps the address of the last node to form a doubly-linked list.

Operations on Linked Lists

- Traversing: To traverse all nodes one by one.
- Insertion: To insert new nodes at specific positions.
- Deletion: To delete nodes from specific positions.
- Searching: To search for an element from the linked list.

Linked List vs Array

ARRAY	LINKED LISTS
1. Arrays are stored in contiguous location.	1. Linked lists are not stored in contiguous location.
2. Fixed in size.	2. Dynamic in size.
3. Memory is allocated at compile time.	3. Memory is allocated at run time.
4. Uses less memory than linked lists.	4. Uses more memory because it stores both data and the address of next node.
5. Elements can be accessed easily.	5. Element accessing requires the traversal of whole linked list.
6. Insertion and deletion operation takes time.	6. Insertion and deletion operation is faster.

Linked List - Applications

1. **Image viewer** – Previous and next images are linked and can be accessed by the next and previous buttons.
2. **Previous and next page in a web browser** – We can access the previous and next URL searched in a web browser by pressing the back and next buttons since they are linked as a linked list.
3. **Music Player –** Songs in the music player are linked to the previous and next songs. So you can play songs either from starting or ending of the list.
4. **GPS navigation systems-** Linked lists can be used to store and manage a list of locations and routes, allowing users to easily navigate to their desired destination.

Recall:

- Linked list – Introduction, need of linked list
- Linked list – Memory representation and allocation
- Linked list – Creation of Node
- Linked list – Types and operations
- Linked list vs Array
- Linked list - Applications