COMPARATIVE ANALYSIS ON SLL, DLL, CLL

Name: Muhammad Yasir Javed
Stud#: K1613614
Section: A

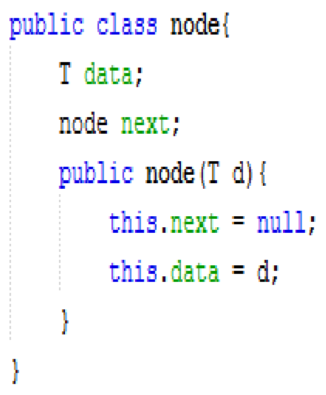
Introduction:

Linked lists are series of nodes joined together using a node. Each node has a linkage pointer to point to either the previous or the next node. We will be discussing linked list comparatively down below.

A linked list basically consists of:

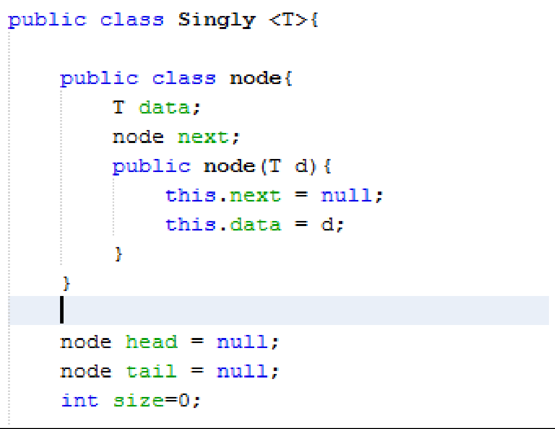
- Node
  Each Node has:
  1. Data
  2. Next
  3. Previous (In case of Doubly Linked List)
- Head (First Node)
- Tail (Last Node)

Structure of a Node:



```
public class node{
    T data;
    node next;
    public node(T d){
        this.next = null;
        this.data = d;
    }
}
```

Structure of a Linked List:



```
public class Singly <T>{

    public class node{
        T data;
        node next;
        public node(T d){
            this.next = null;
            this.data = d;
        }
    }

    node head = null;
    node tail = null;
    int size=0;
```

Comparison:

Singly Linked List	Doubly Linked List	Circular Linked List
Structure Of Singly Linked List: `public class Singly <T>{` `public class node{` `T data;` `node next;` `public node(T d){` `this.next = null;` `this.data = d;` `}` `}` `node head = null;` `node tail = null;` `int size=0;`	Structure Of Doubly Linked List: `public class Doubly<T> {` `public class node<T>{` `T data;` `node next;` `node prev;` `public node(T d){` `this.next = null;` `this.prev = null;` `this.data = d;` `}` `}` `int size=0;` `node head =null;` `node tail = null;`	Circular Linked list entirely similar to the Singly and Doubly Linked list. The interfaces are mirrored but the only minor change is that the **Head** of the list has a link to the **Tail** of the list. i.e. For Singular Circular: Each time when the tail is altered (deleted or inserted) `tail.next = head;` For Doubly Circular: Each time when the tail is altered (deleted or inserted) the code logic is changed to: `tail.next = head;` `tail.prev = beforeTail; //Node before tail` `head.prev = tail;`
1. Only one pointer is maintained in a node of singly list which contains the address of next node in sequence another will keep the address of. Singly linked list uses less memory per node (one pointer)	1. Two pointers are maintained in a node of doubly list, one will keep the address of first previous node and first next node in sequence. Doubly linked list uses More memory per node than Singly Linked list (two pointers)	
2. In singly, we cannot move in backward direction because each in node has next node pointer which facilitates us to move in forward direction.	2. In doubly list, we can move backward as well as forward direction as each node keeps the address of previous and next node in sequence.	
3. During deletion of a node in between the singly list, we will have to keep two nodes address', one is the address of the node to be deleted and second is the node just previous of it.	3. During deletion, we have to keep only one node address i.e the node to be deleted. This node will give the address of previous node automatically.	
4. Complexity of Insertion and Deletion at known position is O (n).	4. Complexity of Insertion and Deletion at known position is O (1).	
5. If we need to save memory in need to update node values frequently and searching is not required, we can use Singly Linked list.	5. If we need faster performance in searching and memory is not a limitation we use Doubly Linked List.	
6. If we know in advance that element to be searched is found near the end of the list (for example name '***yacht***' in a telephone directory), even then singly linked list is traversed sequentially from beginning.	6. In doubly linked list If we know in advance that element to be searched is found near the end of the list(for example name 'Yogesh' in a telephone directory), then the list can traversed from the end thereby saving time	

Applications:

Singly Linked Lists

- The main application of linked list will be waiting a list preparation for doctor's appointment.
  Doctor calls a patient and subsequently another patient is added to the list and the already treated patient leaves (deleted from the list). The list size here keeps changing.

- Items in a Shopping Cart can be processed using a Linked List.

- Linked lists are also used to solve different linear algebra problems. i.e. addition of polynomials etc.

- The simplest and most straightforward is a **train**.
  Train cars are linked in a specific order so that they may be loaded, unloaded, transferred, dropped off, and picked up in the most efficient manner possible.
  For instance, the A food processing plant needs sugar, flour, beans, etc. Just around the next corner might be a paper processing plant that needs chlorine, sulfuric acid, and hydrogen.
  Now, we can stop the train, unload each car of its contents, then let the train go on, but then everything else on the train has to sit while flour is sucked out of the caisson, then the sugar, etc.
  Instead, the cars are loaded on the train in order so that a whole chunk of it can be detached, and the remainder of the train moves on.
  The end of the train is easier to detach than a portion in the middle, and vastly easier than detaching a few cars in one spot, and a few cars in another spot.
  If needed, however, you can insert and remove items at any point in the train.
  Much like a linked list.

- A list of users of a website that need to be emailed some notification

Doubly Linked Lists

- In many operating systems, the thread scheduler (the thing that chooses what processes need to run at which times) maintains a doubly-linked list of all the processes running at any time. This makes it easy to move a process from one queue (say, the list of active processes that need a turn to run) into another queue (say, the list of processes that are blocked and waiting for something to release them). The use of a doubly-linked list here allows each of these splices and rewires to run in time O(1), and the doubly-linked-list structure works well for implementing the scheduler using queues (where you only need to pull things out from the front.)
- A **music player** which has **next** and **previous buttons**.
- Represent a **deck of cards** in a game.
- The **browser cache** which allows you to hit the **BACK-FORWARD** pages.
- **Applications** that have a **Most Recently Used list** (a linked list of file names)
- **Undo-Redo** functionality.

Circular Linked Lists

- Circularly moving escalator**.**
- Railway Station.
- Airport
- Bus Stop (All places where the transportation simultaneously arrives and departed).
- A simple example is keeping track of whose turn it is in a multi-player board game. Put all the players in a circular linked list. After a player takes his turn, advance to the next player in the list. This will cause the program to cycle indefinitely among the players.