Linked List Data Structure and Types

Dr Shantanu Pathak

Linked List

- To store unlimited data elements
- Gives sequential access to stored data elements

- Ex.
- Kites connected to each other
- List of File blocks in Hard Disk
- Iron Chain

Array vs Linked List

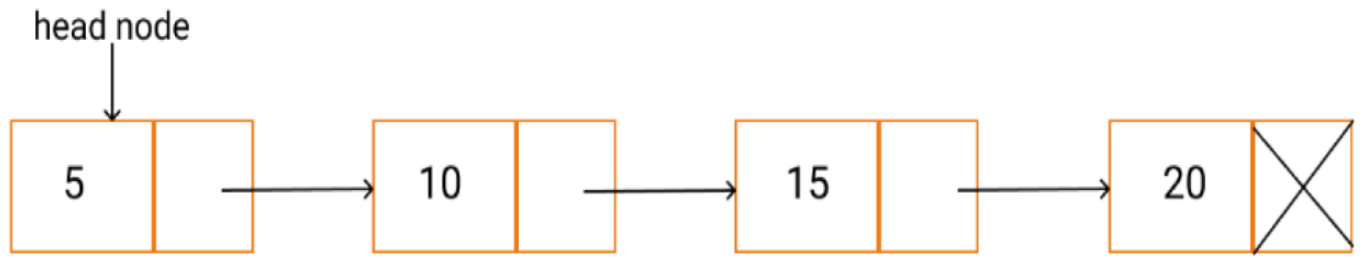
No	Array	Linked List
1	Fixed Size	Dynamic Size
2	Needs Continuous memory locations	Memory may not be continuous
3	Memory Utilization is inefficient Because cannot increase / reduce size	Memory Utilization is efficient because, as per demand can increase / decrease size
4	Accessing element is fast Because, random access to elements	Accessing element is slow Because, sequential access to elements
5	Less memory required Because stores only elements	More memory than Array Because stores element and reference both
6	If array is full then insertion operation takes more time or may be denied	Insertion in any case requires same amount of time

Types of Linked Lists

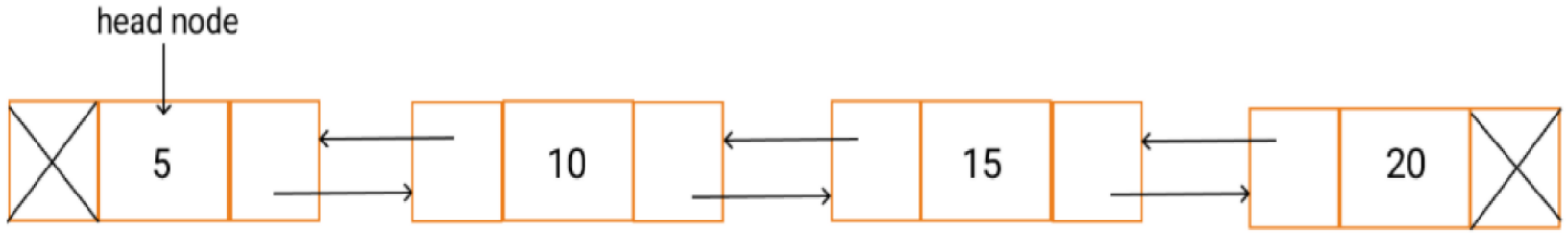
- Singly Linked List
- Doubly Linked List
- Circular Singly Linked List
- Node-based storage with arrays
  - Store linked list in array

Types of Linked Lists

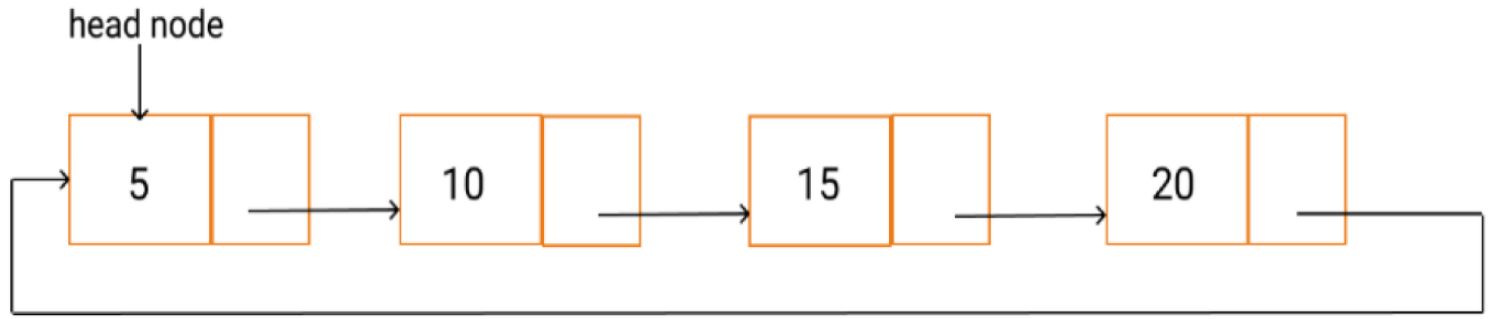
Singly Linked List



Doubly Linked List



Circular Linked List



Singly Linked List Data Structure (D,F,A)

- Domains
  - List of Employee Records
  - List of Book Records
- Functions
  - Insert (same as Insert_at_end)
  - Insert_at_begining
  - Remove (same as Remove_from_end)
  - Remove_from_beginning
  - Display
  - Search
- Axioms / Assumptions
  - Every element contains data and link reference to the next element
  - In Last element of list reference is NULL
  - ‘head’ reference always refers to first element of the linkedlist

Singly Linked List Node

- Create separate class for node
- Contains relationship
- Linked list **contains** node
- Each node
  - Data
  - Next reference
- Data : can be int / char / any object
  - Initialize as per need
- Next reference : reference of Class Node
  - Initialize to null
- Getter / setter in Node class

```
1  public class Node {
2
3      private int data;
4      private Node next;
5
6      public Node() {
7          data = 0;
8          next = null;
9      }
10 }
```

SLL Working

- Insert Node
  - When list empty
  - At the end
- Insert at position
  - At the beginning
  - At the end
  - In between position
  - Invalid position
- Remove Node
  - When list is empty
  - From end
- Remove from position
  - From beginning
  - From end
  - From in between position
  - Invalid position
- Remove Complete Linked List
- Display / Search

SLL memory management

- in creating new node
  - Always set reference to next node as NULL
- in Removing a node
  - Remove the reference to a node, then node is automatically garbage collected
- in Removing Complete Linked List
  - Make the head as NULL complete list will be garbage collected (step by step)
  - NOTE :: In case of very big linked list it may slow down the application

Doubly Linked List Node

- Create separate class for node
- Contains relationship
- Doubly Linked list **contains** node
- Each node
  - Data
  - Prev & Next reference
- Data : can be int / char / any object
  - Initialize as per need
- Prev & Next reference : reference of Class Node
  - Initialize to null
- Getter / setter in Node class

```
1  package dll;
2  public class Node {
3
4      private int data;
5      private Node prev;
6      private Node next;
7
8      public Node() {
9          data = 0;
10         prev=null;
11         next = null;
12     }
```

DLL Working

- Insert Node
  - When list empty
  - At the end
- Insert at position
  - At the beginning
  - At the end
  - In between position
  - Invalid position
- Remove Node
  - When list is empty
  - From end
- Remove from position
  - From beginning
  - From end
  - From in between position
  - Invalid position
- Remove Complete Linked List
- Display / Search

Circular Singly Linked List Node & Working

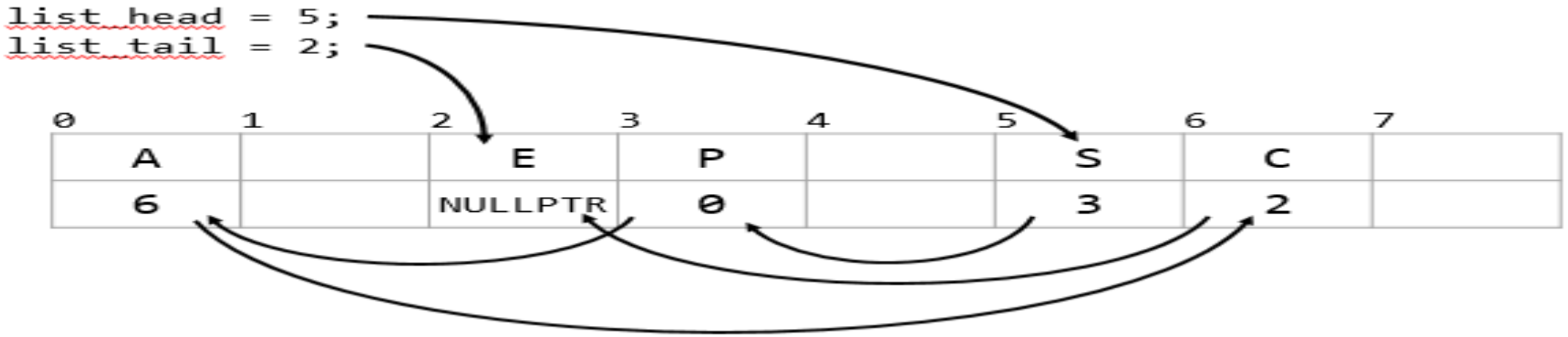
- Node is same as Singly Linked List (SLL)
- All operations like SLL
- Only change in operations::
  - Last node is connected to head
  - So, last node is detected by
  - node.getnext() == head → then it is last node
  - So change operations accordingly

```
1 public class Node {
2     private int data;
3     private Node next;
4
5     public Node() {
6         this.data = 0;
7         next = null;
8     }
```

Node-based storage with arrays

- Linked List drawback
  - New node creation → needs to call OS
  - Adding many nodes at a time → inefficient

- Solution
  - Create an array one time and store nodes in it like Linked List



Linked List Key Notes

- Array vs Linked list -> Imp

- SLL -> as stack
- SLL -> as queue

- **DLL Node** *similar* to **Binary Tree Node**

- CLL -> how to find the end node?