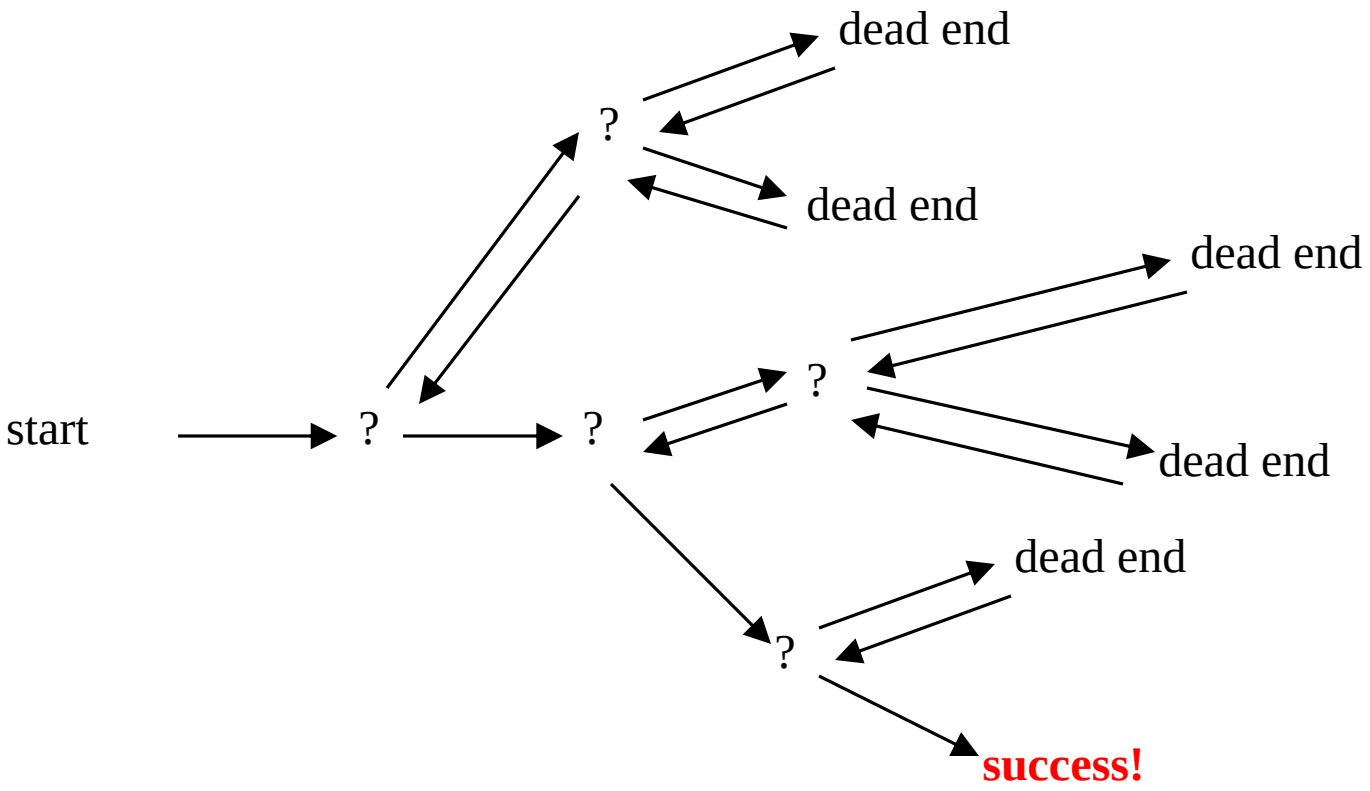
Data Structure

Sep23 : Day 3

Kiran Waghmare

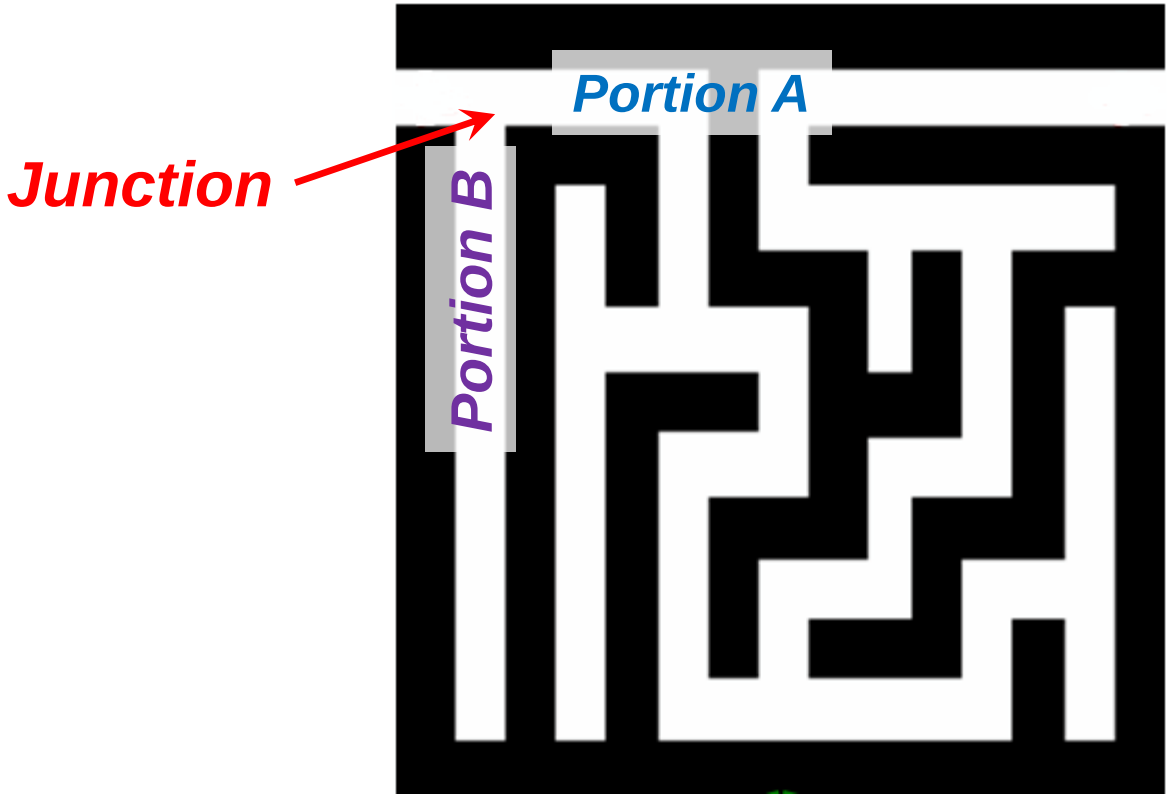
CDAC Mumbai

Backtracking (animation)



Backtracking: Idea

- **Backtracking is a technique used to solve problems with a large search space, by systematically trying and eliminating possibilities.**
- **A standard example of backtracking would be going through a maze.**
  - At some point, you might have two options of which direction to go:



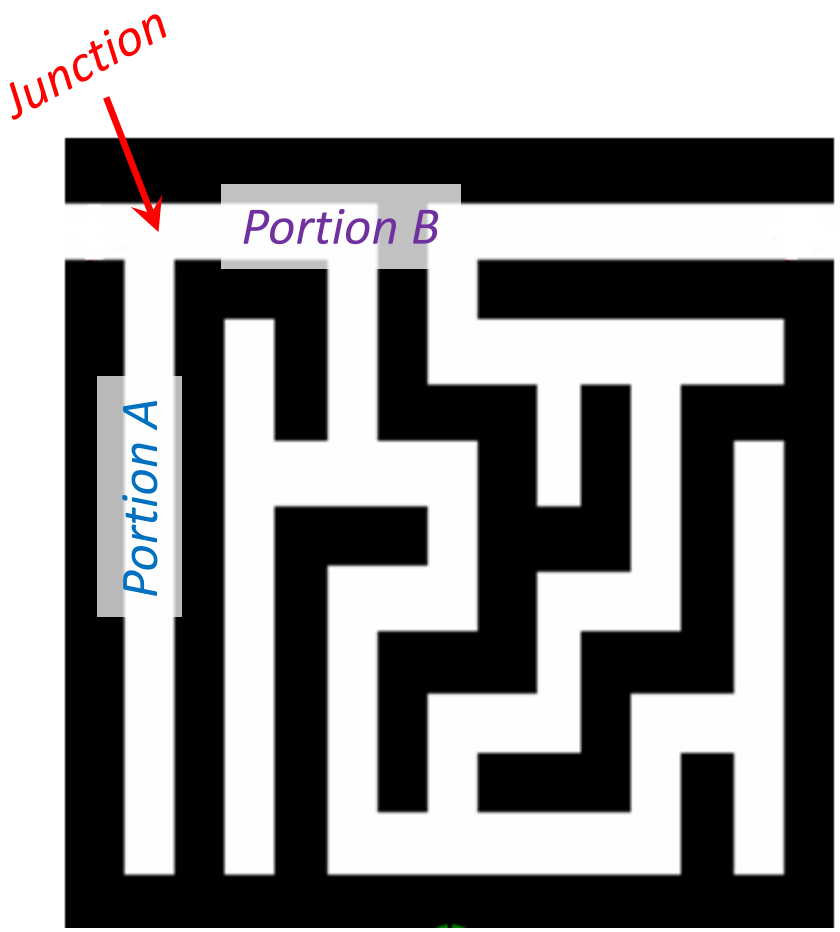
Backtracking

One strategy would be to try going through Portion A of the maze.

If you get stuck before you find your way out, then you *"backtrack"* to the junction.

At this point in time you know that Portion A will *NOT* lead you out of the maze,

so you then start searching in Portion B

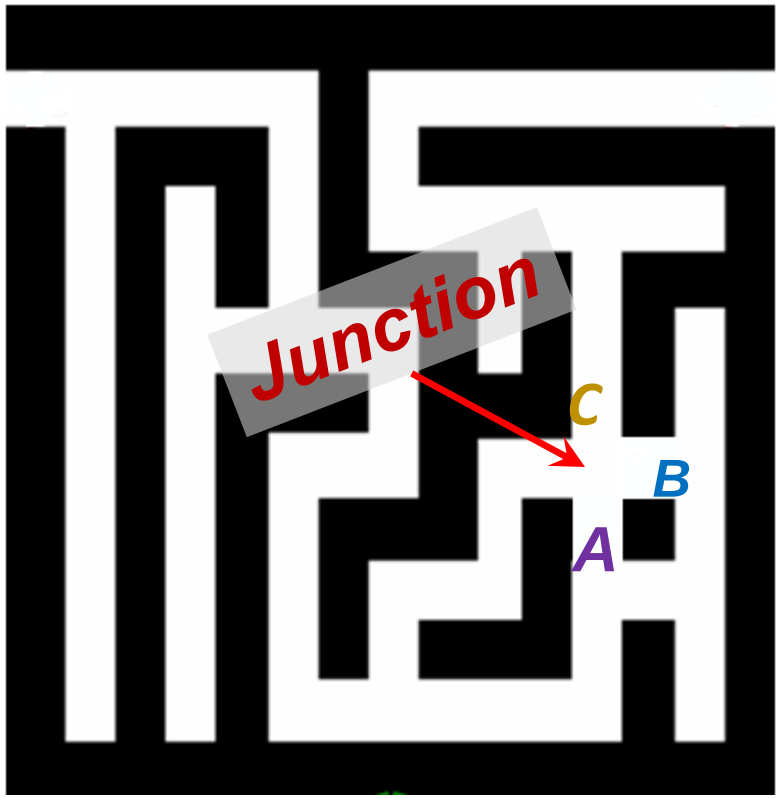


Backtracking

- **Clearly, at a single junction you could have even more than 2 choices.**

- **The backtracking strategy says to try each choice, one after the other,**
  - if you ever get stuck, *"backtrack"* to the junction and try the next choice.

- **If you try all choices and never found a way out, then there IS no solution to the maze.**



Backtracking

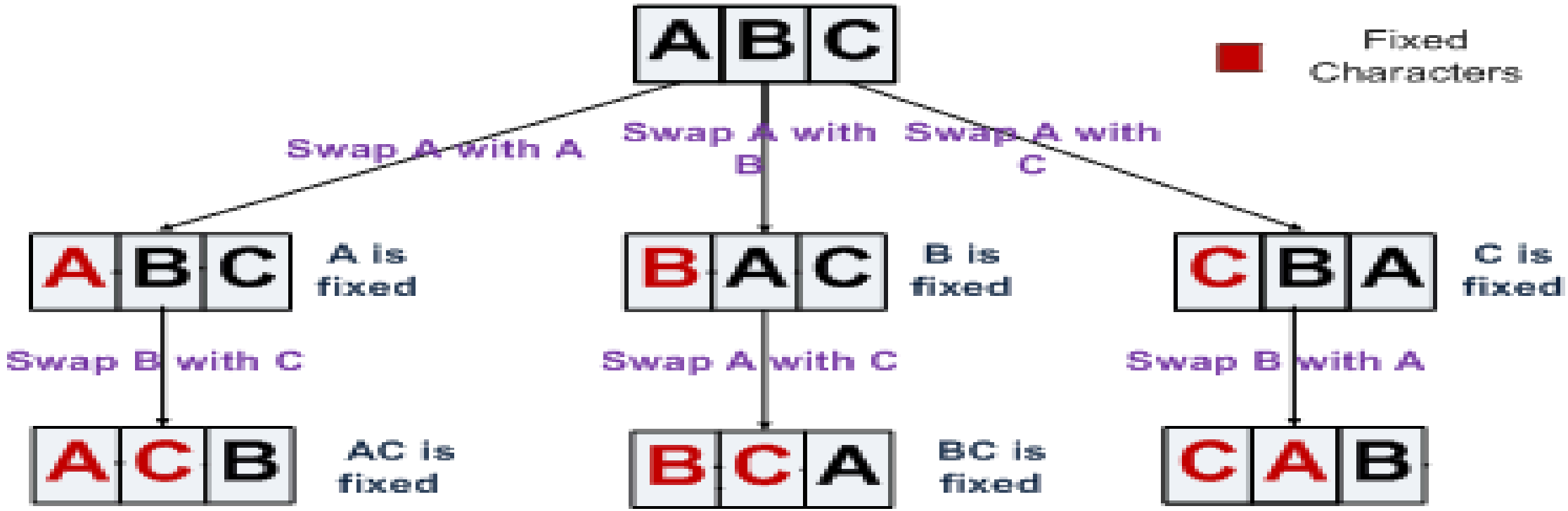
- **Dealing with the maze:**
  - From your start point, you will iterate through each possible starting move.
  - From there, you recursively move forward.
  - If you ever get stuck, the recursion takes you back to where you were, and you try the next possible move.

- **Make sure you don't try too many possibilities,**
  - Mark which locations in the maze have been visited already so that no location in the maze gets visited twice.
  - If a place has already been visited, there is no point in trying to reach the end of the maze from there again.

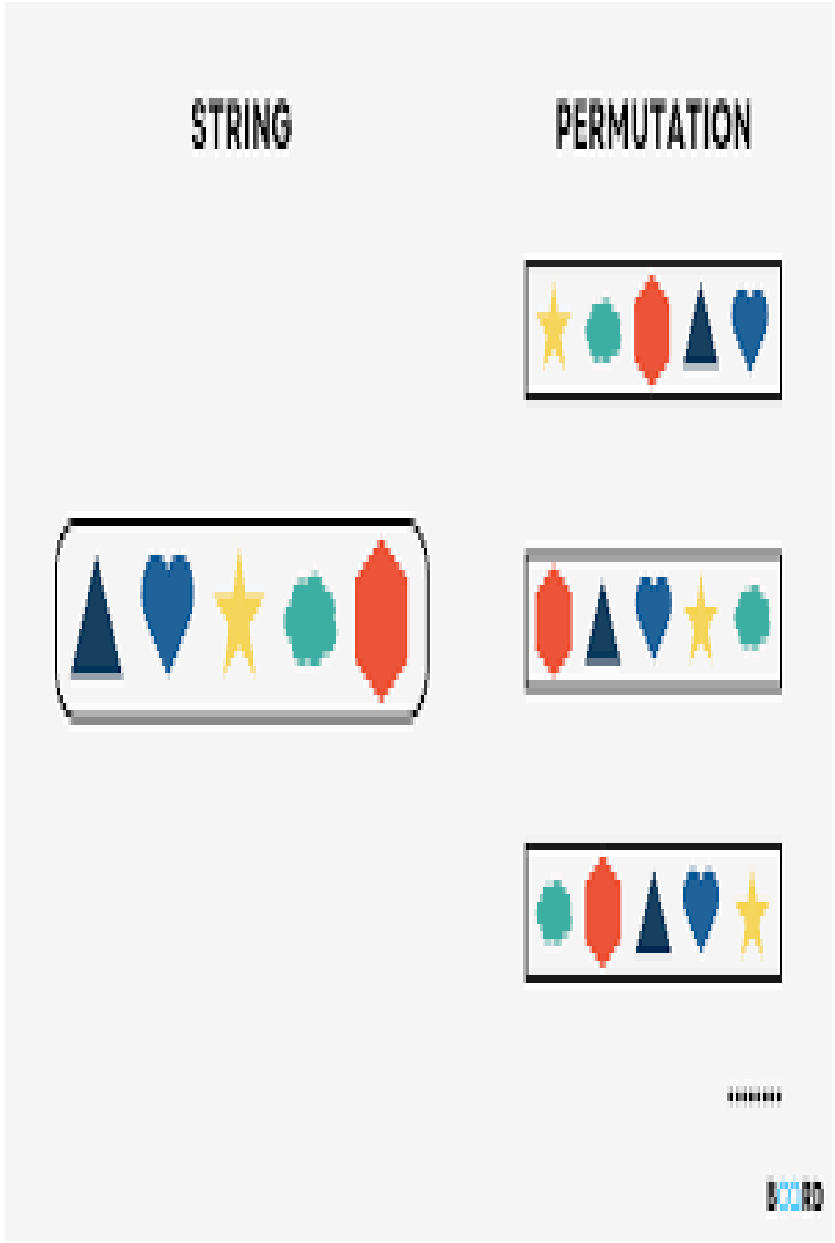
Backtracking

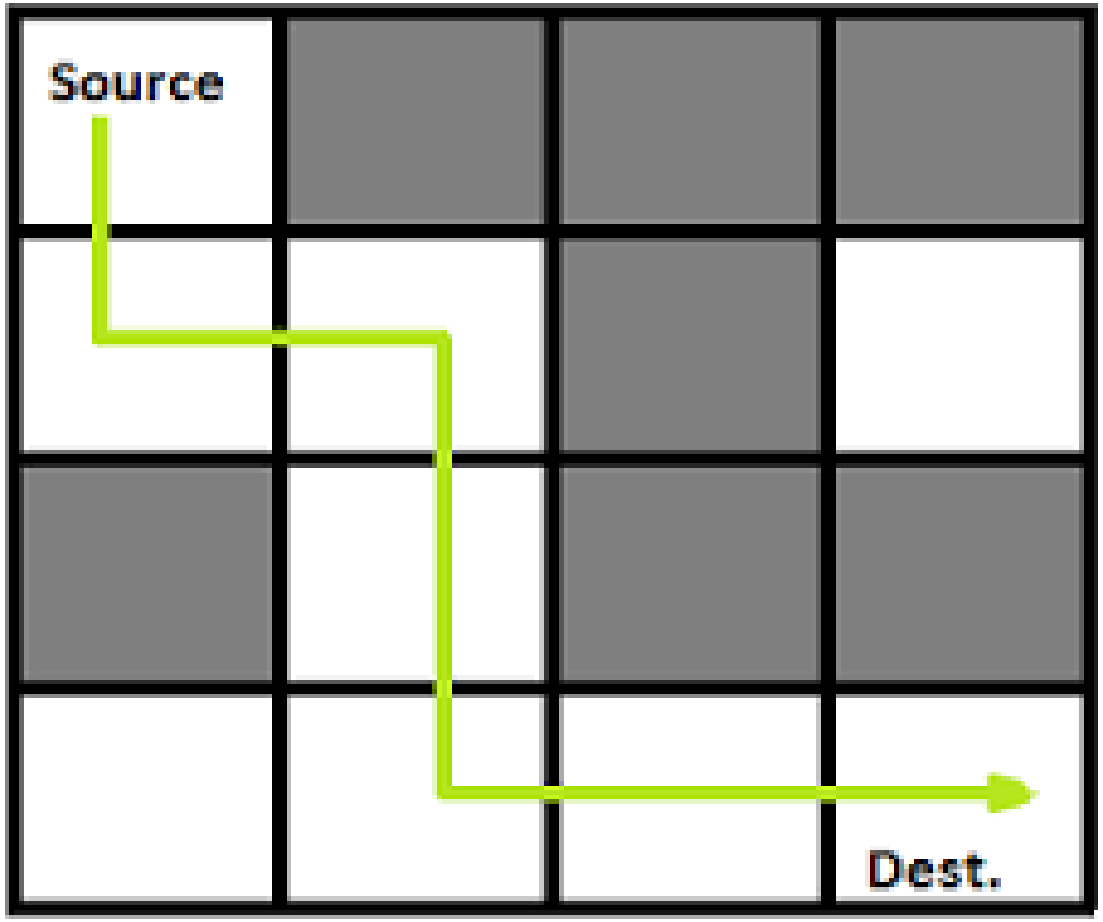
The neat thing about coding up backtracking is that it can be done recursively, without having to do all the bookkeeping at once.

- Instead, the stack of recursive calls does most of the bookkeeping
- (i.e., keeps track of which locations we've tried so far.)

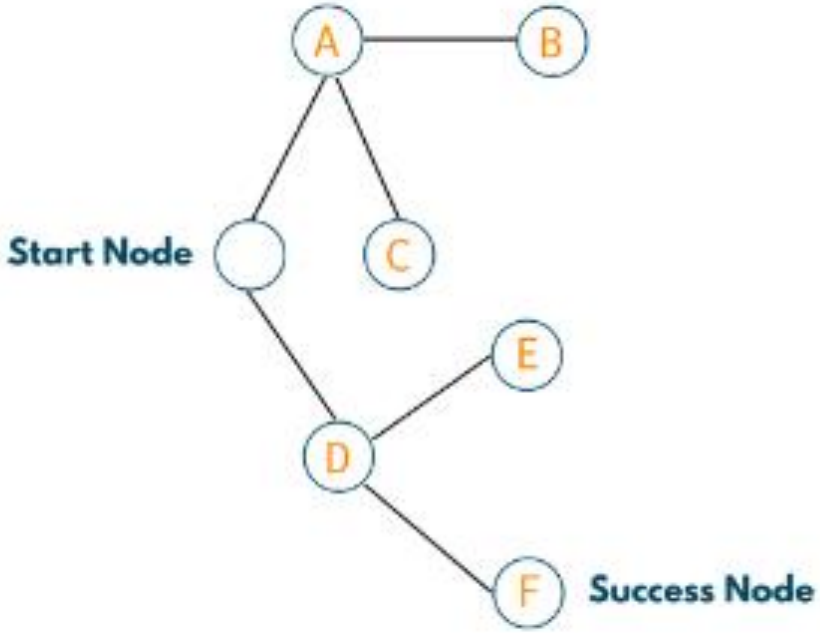


Recursion Tree for Permutations of String "ABC"


STRING
PERMUTATION


Source
Dest.

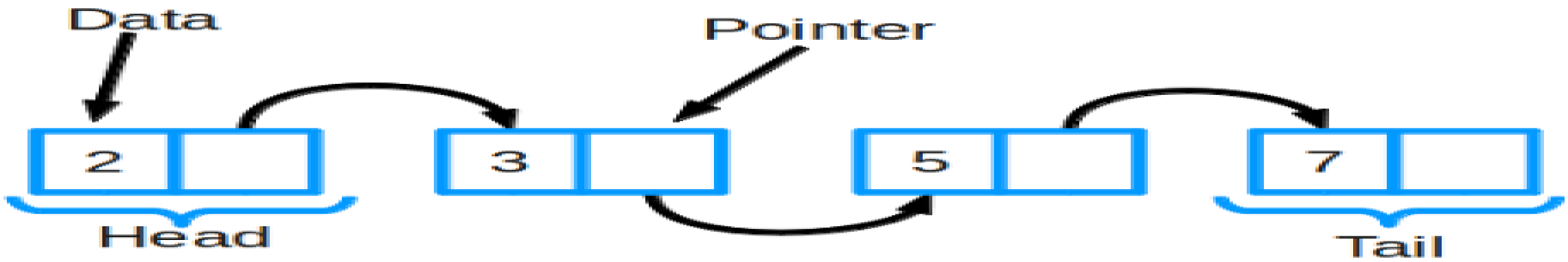
Backtracking Algorithm



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Linked list



Agenda:

-Linked List
-Insertion
-Deletion

-Linked list: It is a sequece of data structure, which are connected via links.
-Sequence of links which contains nodes consist of data and link (reference add next node)

-Link: address of next elements
-Data: value of the current node
-Linked list: list consist of nodes.
-First node: 'head' reference will be connected.
-Last node: link points to 'null' value.

Linked List

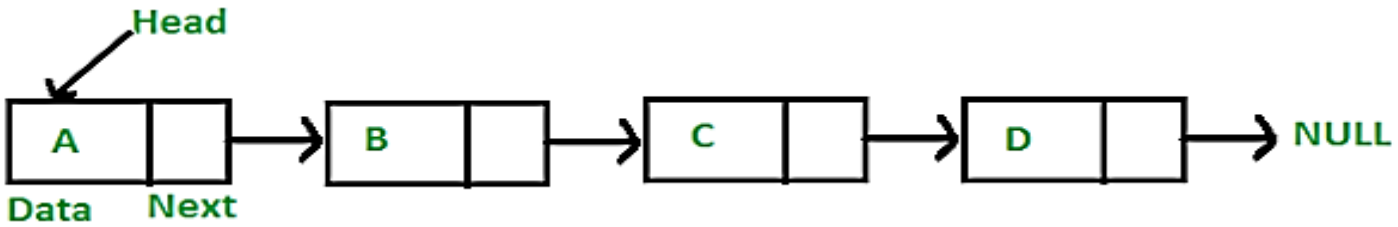
- **A linked list is a sequence of data structures, which are connected together via links.**

- **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.**
- **Following are the important terms to understand the concept of Linked List.**

1. **Link** – Each link of a linked list can store a data called an **element**.
2. **Next** – Each link of a linked list contains a link to the next link called **Next.**
3. **LinkedLis**t – A Linked List contains the **connection link** to the first link called **First**.

Linked List Representation

- **Linked list can be visualized as a chain of nodes, where every node points to the next node.**



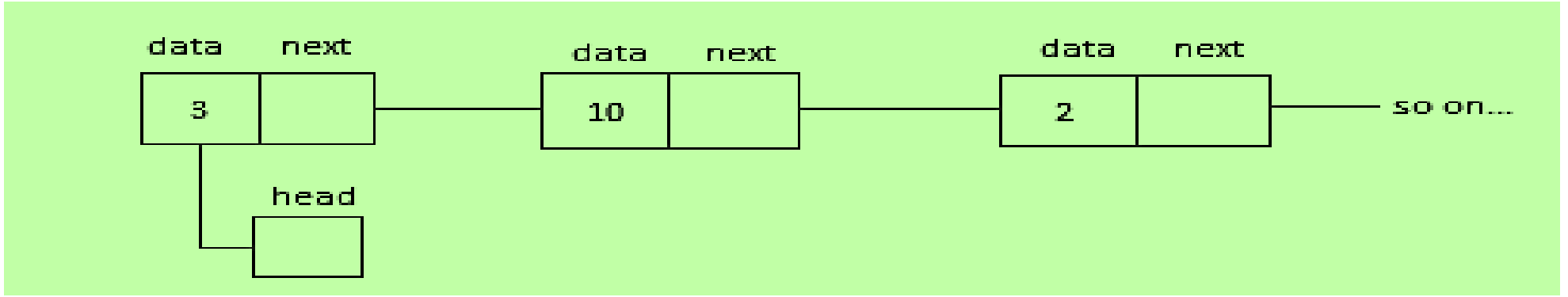
- **As per the above illustration, following are the important points to be considered.**

1. Linked List contains a **link element** called **first.**
2. Each link carries a **data field(s)** and a **link field** called **next.**
3. Each link is **linked with its next link** using its **next link.**
4. **Last link carries a link as null** to mark the end of the list.

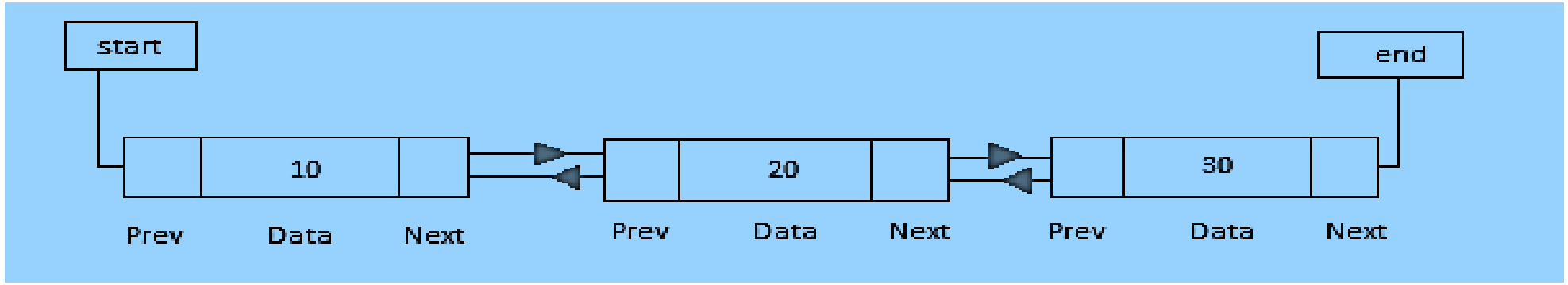
Types of Linked List

- **Following are the various types of linked list.**
  1. **Simple Linked List** – Item navigation is forward only.
  2. **Doubly Linked List** – Items can be navigated forward and backward.
  3. **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.

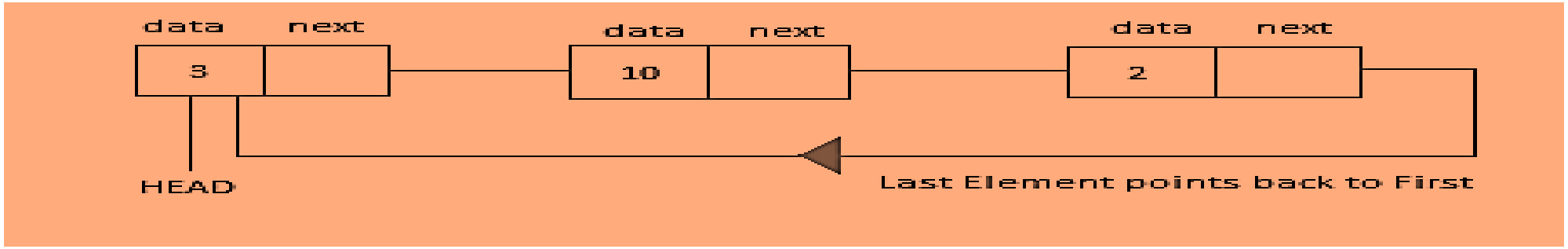
- **Simple Linked List**

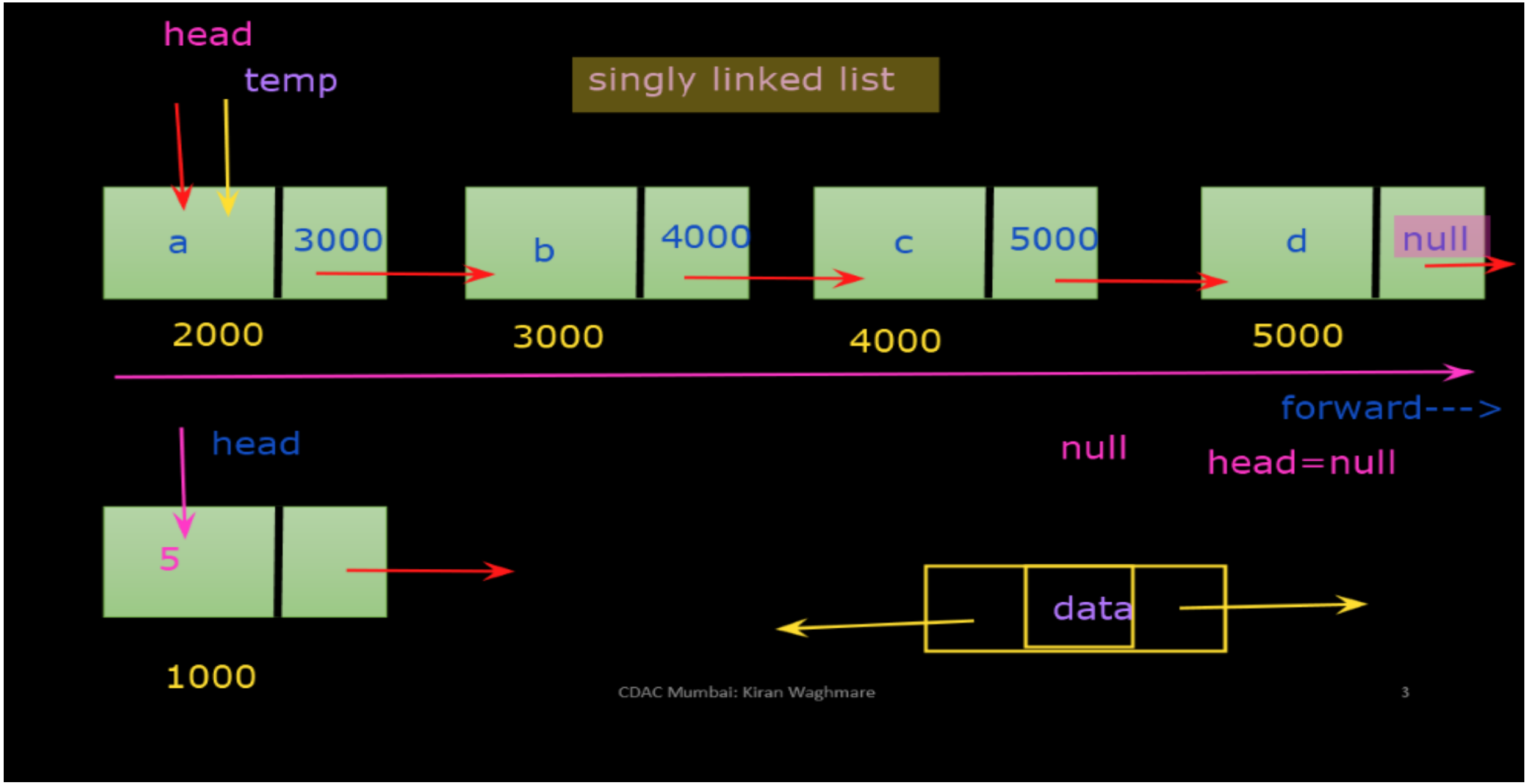


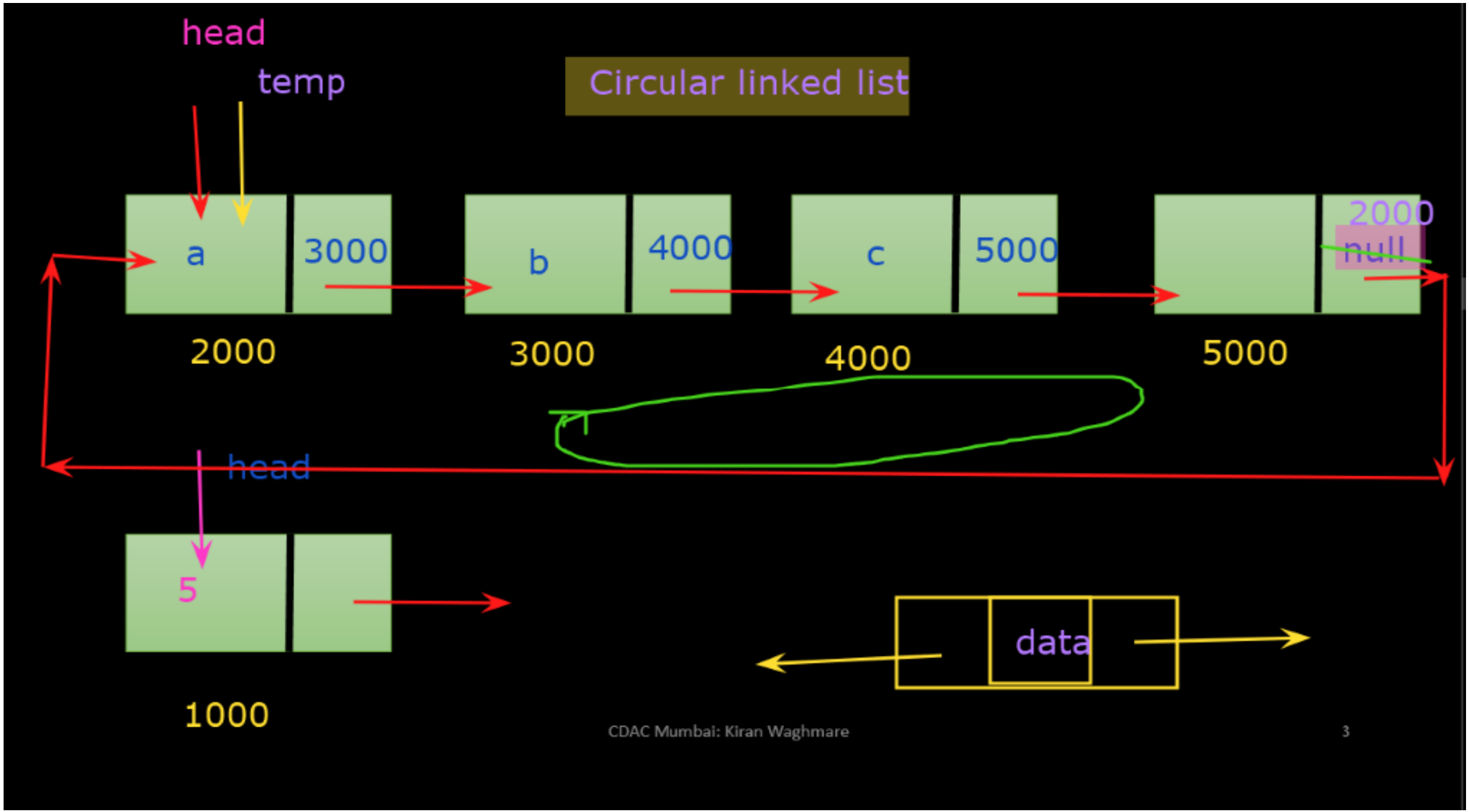
- **Doubly Linked List**

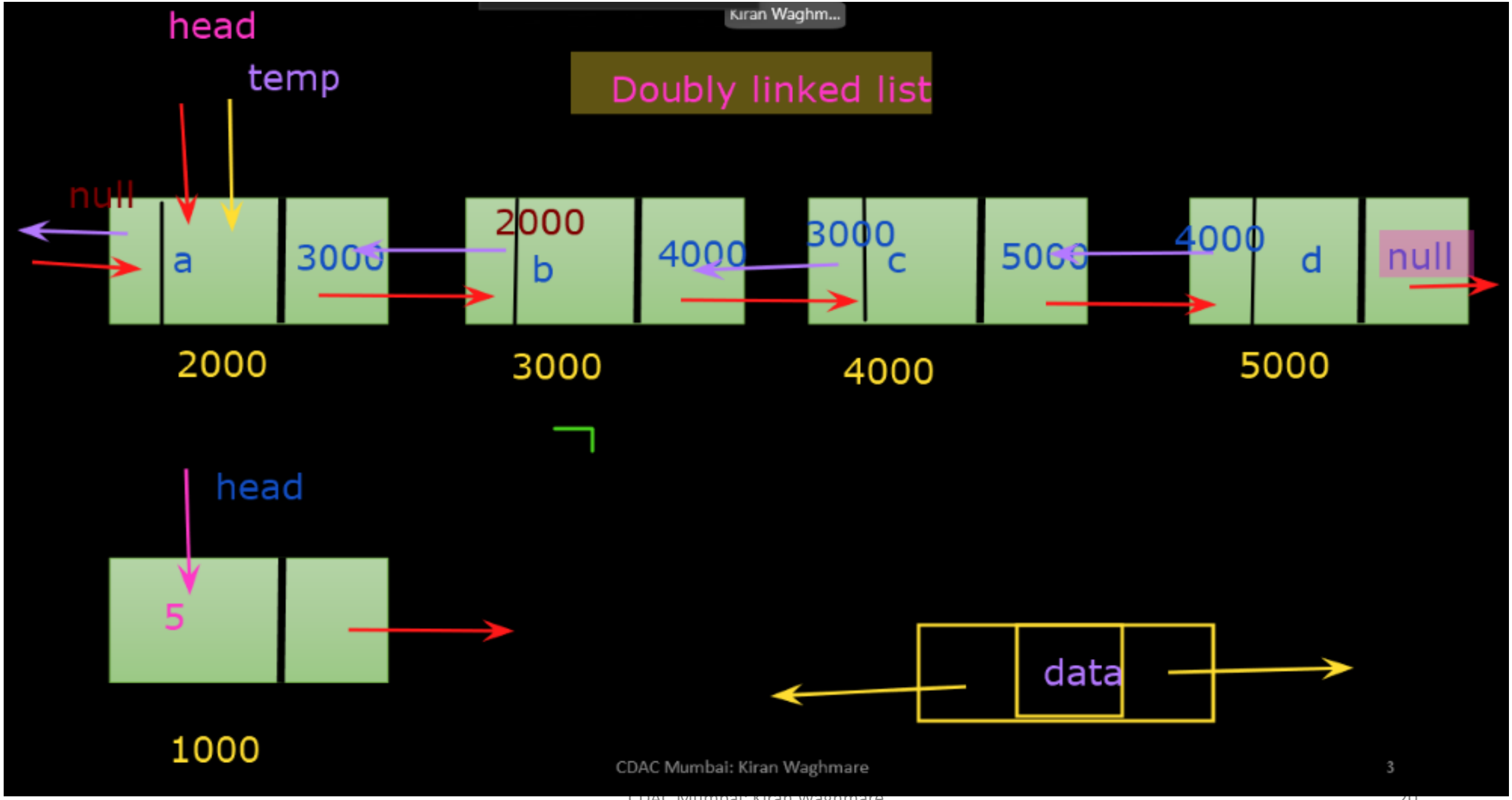


- **Circular Linked List**



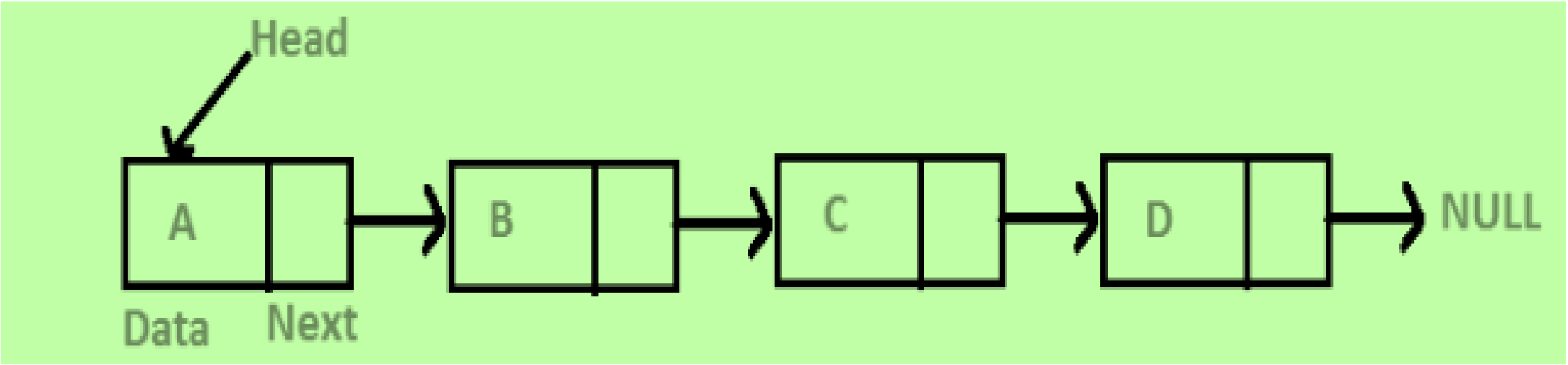

head
temp
singly linked list
a
3000
b
4000
c
5000
d
null
2000
3000
4000
5000
forward--->
head
null
head=null
5
1000
data
CDAC Mumbai: Kiran Waghmare
3


head
temp
Circular linked list
a
3000
b
4000
c
5000
2000
null
2000
3000
4000
5000
head
5
1000
data


Kiran Waghm...
head
temp
Doubly linked list
null
a
3000
2000
b
4000
3000
c
5000
4000
d
null
2000
3000
4000
5000
head
5
1000
data

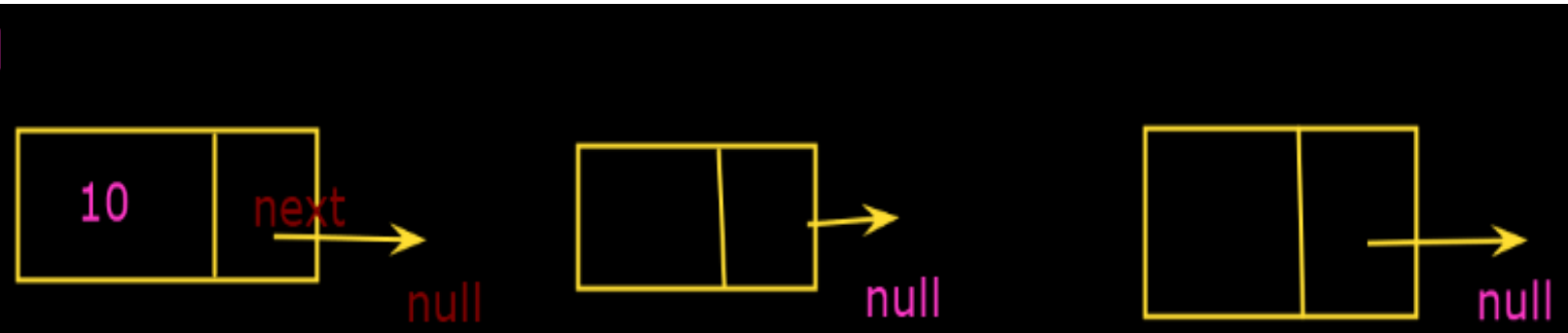
Singly Linked List

- Singly Linked Operations: Insert, Delete, Traverse, search, Sort, Merge



```
static class Node{

    int data;
    Node next;

    Node(int d)
    {
        data=d;
        link=null;
    }
}
```

```
class Linkedlist1{

    Node head;//instance
    static class Node{

        int data;
        Node next;

        Node(int d)
        {
            data=d;
            next=null;
        }
    }

    public static void main(String args[]){

        Linkedlist1 L1 = new Linkedlist1();
        L1.head = new Node(5);
        Node second = new Node(7);
        Node third = new Node(9);
```



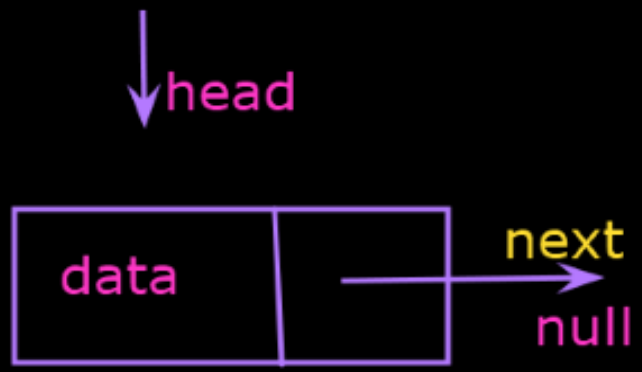
Basic Operations

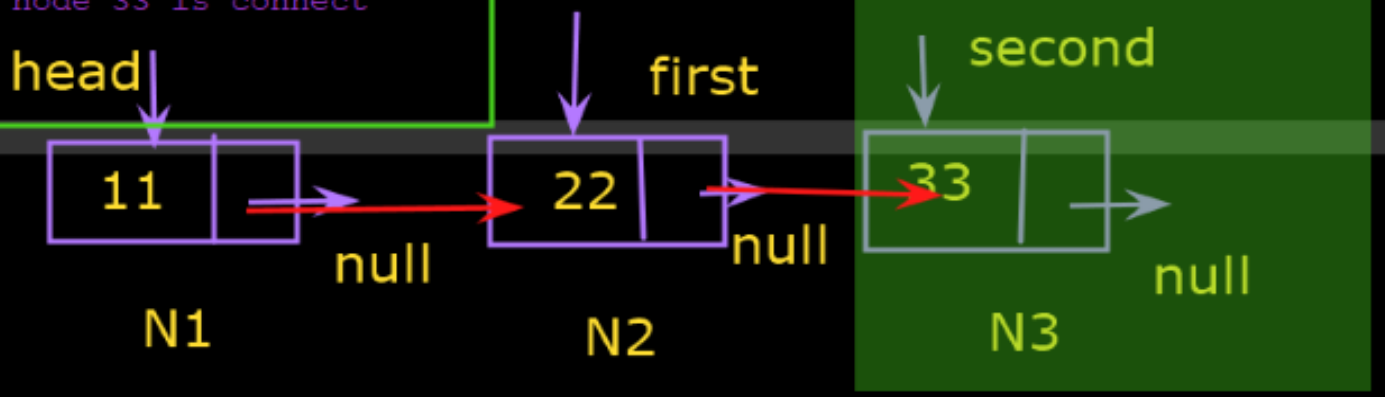
- **Following are the basic operations supported by a list.**
  1. **Insertion** − Adds an element at the beginning of the list.
  2. **Deletion** − Deletes an element at the beginning of the list.
  3. **Display** − Displays the complete list.
  4. **Search** − Searches an element using the given key.
  5. **Delete** − Deletes an element using the given key.

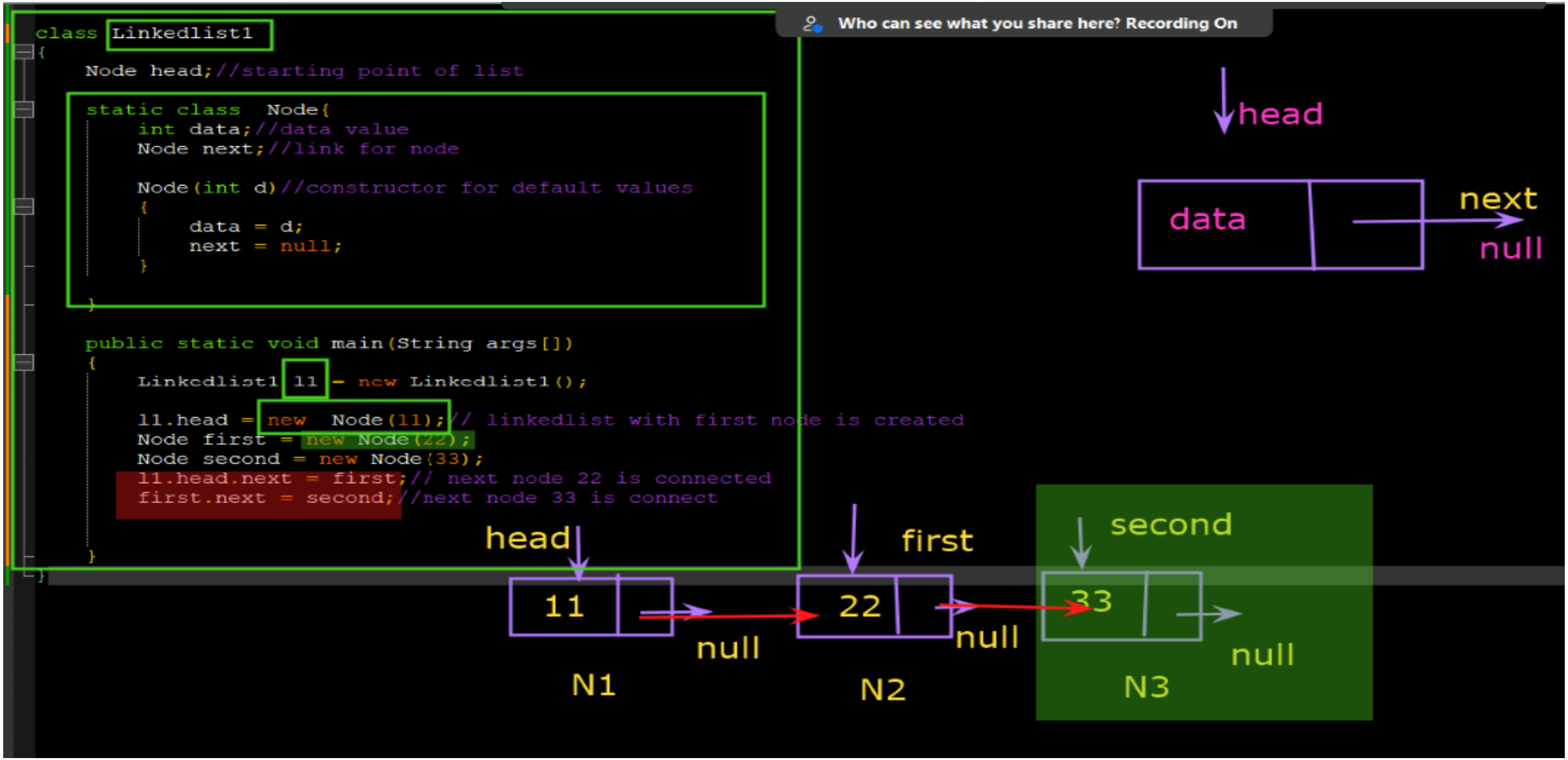
hand

Who can see what you share here? Recording On

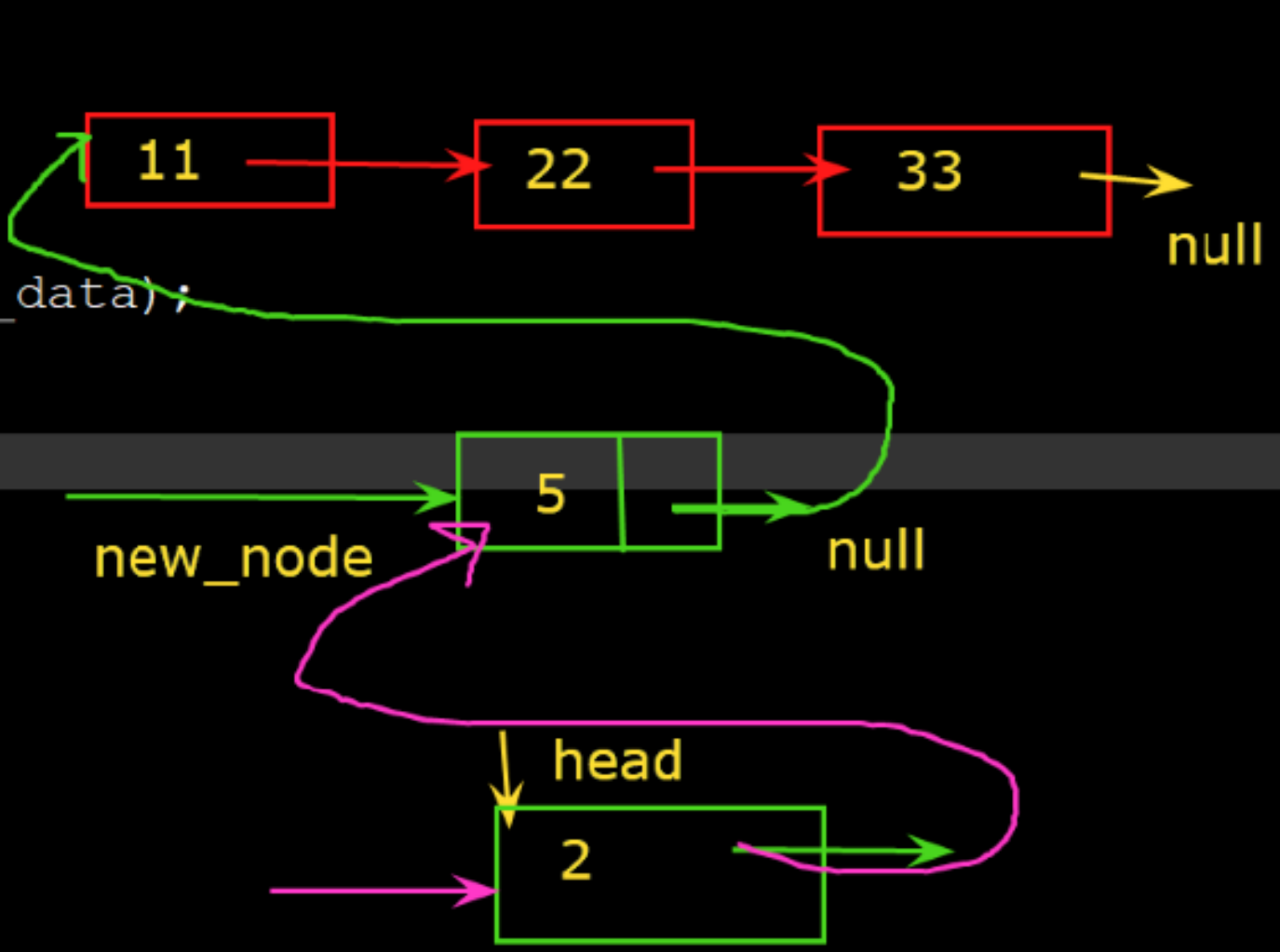
```
class Linkedlist1
{
    Node head;//starting point of list

    static class  Node{
        int data;//data value
        Node next;//link for node

        Node(int d)//constructor for default values
        {
            data = d;
            next = null;
        }

    }

    public static void main(String args[])
    {
        Linkedlist1 ll = new Linkedlist1();

        ll.head = new  Node(11);// linkedlist with first node is created
        Node first = new Node(22);
        Node second = new Node(33);
        ll.head.next = first;// next node 22 is connected
        first.next = second;//next node 33 is connect

    }
}
```






Who can see what you share here? Recording On
class Linkedlist1
{
Node head;//starting point of list
static class Node{
int data;//data value
Node next;//link for node
Node(int d)//constructor for default values
{
data = d;
next = null;
}
}
public static void main(String args[])
{
Linkedlist1 ll = new Linkedlist1();
ll.head = new Node(11);// linkedlist with first node is created
Node first = new Node(22);
Node second = new Node(33);
ll.head.next = first;// next node 22 is connected
first.next = second;//next node 33 is connect
}
}
head
data
next
null
head
first
second
11
22
33
null
null
null
N1
N2
N3

```
Insertion at the begining:
---------------------------

public void insert(int new_data)
{
    Node new_node = new Node(new_data);
    new_node.next = head;
    head = new_node

}
```



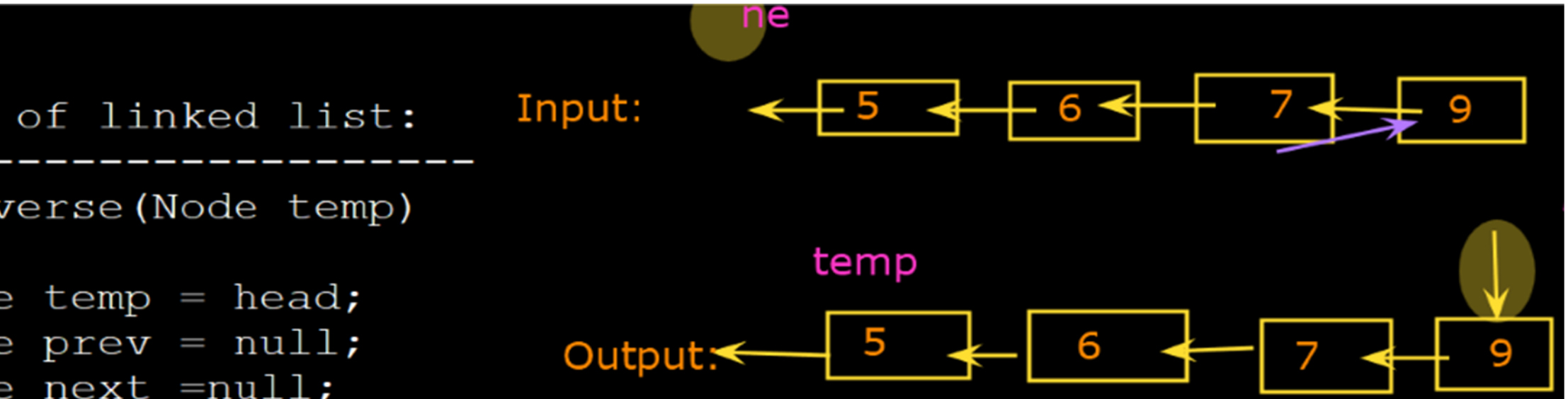
Count no of nodes in linked list:

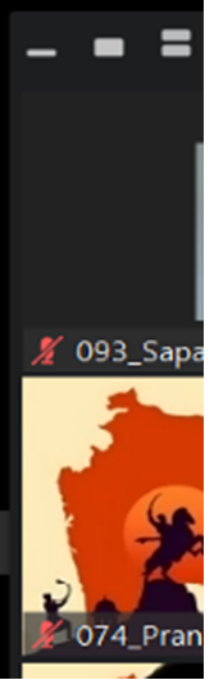
```
int count()
{
    Node temp = head;
    int c=0;

    while(temp != null)
    {
        c++;
        temp=temp.next;

    }
    return c;

}
```

c=0

O(n)

```
}

Reverse of linked list:
-------------------------
Node reverse(Node temp)
{
    Node temp = head;
    Node prev = null;
    Node next =null;

    while(temp != null )
    {
        next = temp.next;
        temp.next = prev;
        prev = temp;
        temp=next;
    }
    head=prev;
    retrun head;

}
```





Problem Statement 1 : Delete a Linked List node at a given position.

Given a singly linked list and a position, delete a linked list node at the given position.

Example:

Input: position = 1, Linked List = 18->12->13->11->17
Output: Linked List = 18->13->11->17

Input: position = 0, Linked List = 98->24->32->17->74
Output: Linked List = 24->32->17->74

Program for Nth node from the end of a Linked List

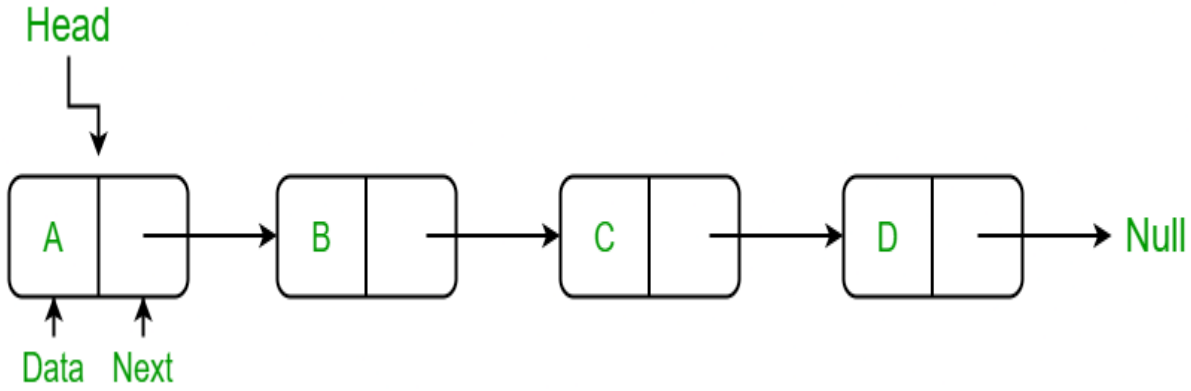
Given a Linked List and a number N, write a function that returns the value at the Nth node from the end of the Linked List.

Linked-List

Examples:

Input: 1 -> 2 -> 3 -> 4, N = 3
Output: 2

Input: 35 -> 15 -> 4 -> 20, N = 4
Output: 35



Thanks