Pavan K
RVCE24MCA092

To Reverse a linked list (both iterative & Recursive)

* To reverse a linked list iteratively, we need to modify the next pointers of each node so that they point in the opposite direction. This can be achieved using three pointer.

1. prev - keeps track of previous node
2. curr - Represents the current node being processed
3. next - stores the next node in the original list to avoid losing track.

Steps to Reverse the linked list

* Set prev to Null
* Assign curr to the head of the linked list
* Keep next as null initially.

Iterate through the list

* Save the next node : [next = curr -> next]
* Reverse the link : [curr -> next]
* move the prev pointer forward : [prev = curr]
* Move curr to the next node : [curr = next]

[Repeat Until curr Becomes Null]

* when curr becomes null, previous (prev) becomes will be at the new head of the reversed list.

Tracing -

Head → [1] → [2] → [3] → [4] → [5] → Null

∴ This is our original linked list

Initially, pointers are

prev = null

curr = Head (1)

next = null.

Step - by Step Reversal Process

Step 1 :- Reverse the first node

* Store : next = curr.next (next = 2)
* Reverse the link : curr.next = prev (1 → Null)
* Move prev & curr ahead.

prev = [1] → Null, curr = [2] → [3] → [4] → [5] → null

Step 2 :- Reverse the second node

prev = [1] ← [2] curr = [3] → [4] → [5] → null

Step 3 :- Reverse the Third node

prev = [1] ← [2] ← [3] curr = [4] → [5] → null

Step 4 :- Reverse the fourth node

prev = [1] ← [2] ← [3] ← [4]

Step 5 :- Reverse the last node

prev = [1] ← [2] ← [3] ← [4] ← [5] curr = Null

Now, the curr is null, & prev is pointing to new head of reversed list.

Reversed ~~List~~ linked list

Head -> [5] -> [4] -> [3] -> [2] -> [1] -> Null

Done with Iterative reverse.

Now Reverse the output with Recursive.

Steps :-

1 :- Base case : If the list is empty (head == Null) or has only one node (head.next == null) return head as it already reversed.]

2 :- Recursive steps :-

* call ~~recursion~~ reverseRecursive (head.next) to reverse

* Adjust pointers :

* head.next.next = head (reverse link of the current node)

* head.next = null (Break the original link to prevent cycles)

3 :- return new head.

Program :-

```
class LinkedList {

    static class Node {

        int data;

        Node next;

        Node (int data) {

            this.data = data;

            this.next = null;

        }

    }

    Node head;
```

```
public void reverseIterative() {        // Iterative method
    Node prev = null, current = head, next = null;
    while (current != null) {
        next = current.next;            // store next node
        current.next = prev;            // Reverse pointer
        prev = current;                 // Move prev to curr
        current = next;                 // Move curr to next
    }
    head = prev;                        // update head
}
```

Recursive method

```
public node reverseRecursive(Node node) {
    If (node == null || node.next == null) {
        return node;                    // Base Case
    }
    Node newHead = reverseRecursive(node.next);
    node.next.next = node;
    node.next = null;
    return newHead;
}
```

```
public void printList(){
    Node temp = head;
    while (temp != null){
        System.out.print(temp.data + "->");
        temp = temp.next;
    }
    System.out.println("Null");
}

public static void main(String[] args){
    LinkedList list = new LinkedList();
    list.head = new Node(1);
    list.head.next = new Node(2);
    list.head.next.next = new Node(3);
    list.head.next.next.next = new Node(4);
    list.head.next.next.next.next = new Node(5);

    System.out.println("Original List");
    list.printList();

    list.reverseIterative();
    System.out.println("Reversed List (Iterative):");
    list.printList();

    list.head = list.reverseRecursive(list.head);
    System.out.println("Reversed List (Recursive)":);
    list.print();
}
}
```

Output :-

Original List

1 2 3 4 5

Reversed List (Iterative)

5 4 3 2 1

Reversed List (Recursive)

1 2 3 4 5