Introduction to LinkedList, learn about struct, and how is node represented

03 October 2024 21:44

```
class Node {
    int data;        // Data stored in the node
    Node next;       // Reference to the next node in the linked list
    // Constructor
    public Node(int data1, Node next1) {
        this.data = data1;
        this.next = next1;
    }
    // Constructor
    public Node(int data1) {
        this.data = data1;
        this.next = null;
    }
}
```

```
class Solution {
    static Node constructLL(int arr[])
    {
        Node head=null;
        Node tail=null;

        for(int i=0;i<arr.length;i++)
        {
            Node curr=new Node(arr[i]);
            if(head==null)
            {
                head=curr;
                tail=curr;
            }
            else
            {
                tail.next=curr;
                tail=tail.next;
            }

        }
        return head;
    }
}
```

Inserting a node in LinkedList

03 October 2024 21:46

```
class Solution {
   // Function to insert a node at the end of the linked list.
   Node insertAtEnd(Node head, int x)
   {
      if(head==null)
      {
         Node curr=new Node(x);
         return curr;
      }
      Node curr=head;

      while(curr.next!=null)
      {
         curr=curr.next;
      }
      curr.next=new Node(x);
      return head;
      // code here
   }
}
```

Deleting a node in LinkedList

03 October 2024 21:46

```
class Solution {
    public void deleteNode(ListNode node)
    {
        if(node.next==null)
        {
            return;
        }
        node.val=node.next.val;
        node.next=node.next.next;
        return;

    }
}
```

Find the length of the linkedlist [learn traversal]

03 October 2024 21:46

```
class Solution {
   // Function to count nodes of a linked list.
   public int getCount(Node head)
   {
      if(head==null)
      {
         return 0;
      }
      int cnt=0;
      while(head!=null)
      {
         cnt++;
         head=head.next;
      }
      return cnt;
      // code here
   }
}
```

Search an element in the LL

03 October 2024 21:47

```
class Solution {
   static boolean searchKey(int n, Node head, int key)
   {
      while(head!=null)
      {
         if(head.data==key)
         {
            return true;
         }
         head=head.next;
      }
      return false;
   }
}
```

Introduction to DLL, learn about struct, and how is node represented

03 October 2024 21:47

```
class Solution {
    Node constructDLL(int arr[])
    {
        int n=arr.length;

        Node head=null;
        Node tail=null;

        for(int i=0;i<n;i++)
        {
            Node curr=new Node(arr[i]);

            if(head==null)
            {
                head=curr;
                tail=curr;
            }
            else
            {
                tail.next=curr;
                curr.prev=tail;
                tail=tail.next;
            }
        }
        return head;
    }
}
```

```
public class Node {
    public int data;      // Data stored in the node
    public Node next;     // Reference to the next node in the list (forward direction)
    public Node back;     // Reference to the previous node in the list (backward direction)

    // Constructor for a Node with both data, a reference to the next node, and a reference to the previous node
    public Node(int data, Node next, Node back) {
        this.data = data;
        this.next = next;
        this.back = back;
    }

    // Constructor for a Node with data, and no references to the next and previous nodes (end of the list)
    public Node(int data) {
        this.data = data;
        this.next = null;
        this.back = null;
    }
}
```

Insert a node in DLL

03 October 2024 21:47

```
class Solution {
   // Function to insert a new node at given position in doubly linked list.
   Node addNode(Node head, int p, int x)
   {
      if(head==null)
      {
         Node curr=new Node(x);
         return curr;
      }
      Node ans=head;
      //Node curr=head;
      while(ans!=null&&p>0)
      {
         ans=ans.next;
         p--;
      }
      Node curr=new Node(x);

      if(ans.next!=null)
      {
         curr.next=ans.next;
         ans.next.prev=curr;
      }

      curr.prev=ans;
      ans.next=curr;
      //ans=ans.next;
      return head;
      // Your code here
   }
}
```

Delete a node in DLL**

03 October 2024 21:47

```
class Solution {
   public Node deleteNode(Node head, int x)
   {
      Node curr=head;
      if(x==1)    //what if x==1 and root,next==null
      {
         if(head.next==null)
         {
            return null;
         }
         else
         {
            curr.next.prev=null;
            head=curr.next;
            curr.next=null;
            return head;
         }

      }
      curr=head;

      int i=1;
      while(i!=x)
      {
         curr=curr.next;
         i++;
      }
      curr.prev.next=curr.next;
      if(curr.next!=null)
      {
         curr.next.prev=curr.prev;
      }
      return head;
   }
}
```

Reverse a DLL**

03 October 2024 21:47

```
class Solution {
   public static Node reverseDLL(Node head)
    {
        if (head == null || head.next == null)
            return head;
        Node curr = head;
        Node temp = null;
        while (curr != null)
        {
            curr.prev = curr.next;
            curr.next = temp;
            temp=curr;
            curr = curr.prev;
        }
      return temp;
    }
}
```

Middle of a LinkedList [TortoiseHare Method]

05 October 2024 10:57

```
class Solution {
    public ListNode middleNode(ListNode head)
    {
        ListNode slow=head;
        ListNode fast=head;
        while(fast!=null&&fast.next!=null)
        {
            fast=fast.next.next;
            slow=slow.next;
        }
        return slow;

    }
}
```

Reverse a LinkedList [Iterative]

05 October 2024 10:58

```
class Solution {
    public ListNode reverseList(ListNode head)
    {
        ListNode curr=head;
        ListNode prev=null;
        ListNode next;
        while(curr!=null)
        {
            next=curr.next;
            curr.next=prev;
            prev=curr;
            curr=next;
        }
        return prev;

    }
}
```

Reverse a LL [Recursive]

05 October 2024 10:58

```
public ListNode reverseList(ListNode head)
    {
        if(head==null)||head.next==null)
        {
            return head;
        }
        ListNode curr=reverseList(head.next);
        head.next.next=head;
        head.next=null;
        return curr;

    }
```

Detect a loop in LL

05 October 2024 10:58

#CORRECT ANS

```
public boolean hasCycle(ListNode head)
    {
        ListNode slow=head;
        ListNode fast=head;
        while(fast!=null && fast.next!=null)
        {
            slow=slow.next;
            fast=fast.next.next;
            if(slow==fast)
            {
                return true;
            }

        }

        return false;

    }
```

#INCORRECT ANS WHY?????

```
public boolean hasCycle(ListNode head)
    {
        ListNode slow=head;
        ListNode fast=head;
        while(fast!=null && fast.next!=null)
        {
            if(slow==fast) //BC ye always true hoga
            {
                return true;
            }
            slow=slow.next;
            fast=fast.next.next;
        }

        return false;

    }
```

Find the starting point in LL

05 October 2024 10:58

```
public ListNode detectCycle(ListNode head)
    {
        ListNode slow=head;
        +++12.
fast=head;
        while(fast!=null && fast.next!=null)
        {
            slow=slow.next;
            fast=fast.next.next;
            if(slow==fast)
            {
                break;
            }
        }
        ListNode curr=head;
        if (fast == null || fast.next == null)
         return null;
        while (curr != slow)
        {
            curr = curr.next;
            slow = slow.next;
        }
        return curr;
    }
```

Length of Loop in LL

05 October 2024 10:58

```
public int countNodesinLoop(Node head)
  {
    Node slow=head;
    Node fast=head;

    while(fast!=null && fast.next!=null)
    {
      slow=slow.next;
      fast=fast.next.next;
      if(slow==fast)
      {
        break;
      }
    }
    if(fast==null||fast.next==null)
    {
      return 0;
    }
    int cnt=1;
    fast=fast.next;
    while(fast!=slow)
    {
      fast=fast.next;
      cnt++;
    }
    return cnt;
    // Add your code here.
  }
```

Check if LL is palindrome or not

05 October 2024 10:58

```
class Solution {
    ListNode demo;
    public boolean find(ListNode head)
    {
        if(head==null)
        {
            return true;
        }

        boolean flag=find(head.next);

        if(flag==false)
        {
            return false;
        }
        else if(demo.val!=head.val)
        {
            return false;
        }
        else
        {
            demo=demo.next;
        return true;
        }

    }
    public boolean isPalindrome(ListNode head)
    {
        if(head==null||head.next==null)
        {
            return true;
        }
        demo=head;
        return find(head);

    }
}
```

```
class Solution {
    ListNode demo;
    int cnt=0,fl=0;
    public void find(ListNode head)
    {
        if(head==null)
        {
            cnt=cnt/2;
            return;
        }
        cnt++;
        find(head.next);
        if(cnt==0)
        {
            return;
        }
        if(fl==1)
```

```
        {
            return;
        }

        if(head.val==demo.val)
        {
            cnt--;
            demo=demo.next;
        }
        else
        {
            fl=1;
        }
        return;

    }
    public boolean isPalindrome(ListNode head)
    {
        if(head==null||head.next==null)
        {
            return true;
        }
        demo=head;
        find(head);
        if(cnt==1||fl==1){
            return false;
        }
        return true;

    }
}
```

Segrregate odd and even nodes in LL

05 October 2024 10:58

Odd even index ke Basis pe

```
public ListNode oddEvenList(ListNode head)
{
    ListNode odd=head;
    if(head==null||head.next==null)
    {
        return head;
    }
    ListNode even=head.next;
    ListNode event=head.next;
    while(odd.next!=null&&odd.next.next!=null)
    {
        odd.next=odd.next.next;
        odd=odd.next;
        event.next=event.next.next;
        event=event.next;
    }
    odd.next=even;
    return head;

}
```

Remove Nth node from the back of the LL

05 October 2024 10:58

```
public ListNode removeNthFromEnd(ListNode head, int n)
    {
        int cnt=1;
        ListNode curr=head;
        while(curr.next!=null)
        {
            cnt++;
            curr=curr.next;
        }

        n=cnt-n;
        if(n==0)
        {
            return head.next;
        }
        if(n==1)
        {
            head.next=head.next.next;
            return head;
        }
        if(n<=0)
        {
            return head;
        }
        cnt=1;
        curr=head;

        ListNode prev=null;
        while(n>0)
        {
            prev=curr;
            curr=curr.next;
            n--;
        }
        prev.next=curr.next;;
        return head;

    }
```

```
public ListNode removeNthFromEnd(ListNode head, int n) {
   ListNode dummy = new ListNode(0);  // Dummy node to handle edge cases easily
   dummy.next = head;

   ListNode fast = dummy;
   ListNode slow = dummy;

   // Move fast pointer n steps ahead
   for (int i = 0; i < n; i++) {
      fast = fast.next;
   }

   // Move both fast and slow until fast reaches the end
   while (fast.next != null) {
      fast = fast.next;
```

```
            slow = slow.next;
        }

        // Skip the target node
        slow.next = slow.next.next;

        return dummy.next;  // Return the new head
    }
```

Delete the middle node of LL

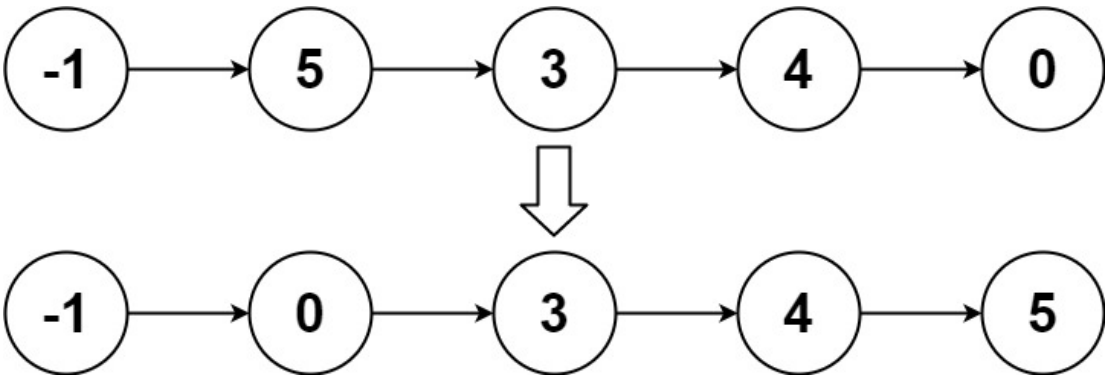
05 October 2024 10:58

```
class Solution {
    public ListNode deleteMiddle(ListNode head)
    {
        if(head==null)
        {
            return head;
        }
        if(head.next==null)
        {
            return null;
        }
        ListNode slow=head;
        ListNode fast=head;
        while(fast!=null&&fast.next!=null)
        {
            slow=slow.next;
            fast=fast.next.next;
        }
        ListNode curr=head;
        while(curr.next!=slow)
        {
            curr=curr.next;
        }
        curr.next=curr.next.next;
        return head;

    }
}
```

Sort LL*****

05 October 2024 10:59




```
public ListNode findMid(ListNode head)
    {
        ListNode slow=head;
        ListNode fast=head;
        while(fast.next!=null&&fast.next.next!=null)
        {
            slow=slow.next;
            fast=fast.next.next;
        }
        return slow;
    }
    ListNode mergeSort(ListNode left,ListNode right)
    {
        ListNode dummy=new ListNode(-1);
        ListNode curr=dummy;
        while(left!=null&&right!=null)
        {
            if(left.val<right.val)
            {
                curr.next=left;
                curr=curr.next;
                left=left.next;
            }
            else
            {
                curr.next=right;
                curr=curr.next;
                right=right.next;
            }
        }
        if(left!=null)
        {
            curr.next=left;
        }
        else
        {
            curr.next=right;
        }


        return dummy.next;
```

```
}
public ListNode sortList(ListNode head)
{
    if(head==null||head.next==null)
    {
        return head;
    }
    ListNode mid=findMid(head);
    ListNode right=mid.next;
    mid.next=null;
    ListNode left= head;
    left=sortList(head);
    right=sortList(right);
    return mergeSort(left,right);
}
```

Sort a LL of 0's 1's and 2's by changing links

05 October 2024 10:59

My Code

```
public static Node sortList(Node head)
   {
      Node dummy1=new Node(-1);
      Node z=dummy1;
      Node dummy2=new Node(-1);
      Node o=dummy2;
      Node dummy3=new Node(-1);
      Node t=dummy3;
      Node curr=head;
      while(curr!=null)
      {
         if(curr.data==0)
         {
            z.next=curr;
            z=z.next;
         }
         else if(curr.data==1)
         {
            o.next=curr;
            o=o.next;
         }
         else
         {
            t.next=curr;
            t=t.next;
         }
         curr=curr.next;
      }



// Connect the three sublists
      z.next = dummy2.next != null ? dummy2.next : dummy3.next;
      o.next = dummy3.next;
      t.next = null; // important to end the final list
      return dummy1.next; // head of the sorted list

      z.next=dummy2.next;
      o.next=dummy3.next;;
      t.next=null;
      dummy1=dummy1.next;
      dummy2=dummy2.next;
      dummy3=dummy3.next;


      if(dummy1==null)
      {
         if(dummy2==null)
         {
            if(dummy3==null)
            {
               return null;
            }
            else
            {
```

```
                return dummy3;
            }
        }
        else
        {
            return dummy2;
        }
    }
    else
    {
        return dummy1;
    }
    // Write your code here
}
```

```
public static Node sortList(Node head) {
    // Dummy heads for three separate lists: 0s, 1s, and 2s
    Node zeroDummy = new Node(-1), oneDummy = new Node(-1), twoDummy = new Node(-1);
    Node zeroTail = zeroDummy, oneTail = oneDummy, twoTail = twoDummy;

    Node curr = head;

    // Separate nodes into three lists based on value
    while (curr != null) {
        if (curr.data == 0) {
            zeroTail.next = curr;
            zeroTail = zeroTail.next;
        } else if (curr.data == 1) {
            oneTail.next = curr;
            oneTail = oneTail.next;
        } else {
            twoTail.next = curr;
            twoTail = twoTail.next;
        }
        curr = curr.next;
    }

    // Connect the three sublists
    zeroTail.next = oneDummy.next != null ? oneDummy.next : twoDummy.next;
    oneTail.next = twoDummy.next;
    twoTail.next = null; // important to end the final list

    return zeroDummy.next; // head of the sorted list
}
```

Find the intersection point of Y LL

05 October 2024 10:59

```
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB)
    {
        int cnt1=0;
        int cnt2=0;
        ListNode head=headA;
        while(head!=null)
        {
            head=head.next;
            cnt1++;
        }
        head=headB;
        while(head!=null)
        {
            head=head.next;
            cnt2++;
        }
        if(cnt1>cnt2)
        {
            int k=cnt1-cnt2;
            ListNode curr=headA;
            while(k>0)
            {
                curr=curr.next;
                k--;
            }
            ListNode temp=headB;
            while(curr!=temp)
            {
                curr=curr.next;
                temp=temp.next;
            }
            return curr;
        }
        else
        {
            int k=cnt2-cnt1;
            ListNode curr=headB;
            while(k>0)
            {
                curr=curr.next;
                k--;
            }
            ListNode temp=headA;
            while(curr!=temp)
            {
                curr=curr.next;
                temp=temp.next;
            }
            return curr;
        }

    }
}
```

Add 1 to a number represented by LL******

05 October 2024 10:59

```
int helper(ListNode head)
    {
        if(head==null)
        {
            return 1;
        }

        int carry=helper(head.next);
        head.val=head.val+carry;
        if(head.val<10)
        {
            return 0;
        }
        head.val=0;
        return 1;
    }
    public ListNode addOne(ListNode head)
    {
        int carry=helper(head);

        if(carry==1)
        {
            ListNode curr=new ListNode(1);
            curr.next=head;
            return curr;
        }
        return head;

    }
```

Add 2 numbers in LL

05 October 2024 10:59

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2)
    {
        ListNode dummy = new ListNode(0); // creating an dummy list
        ListNode curr = dummy; // intialising an pointer
        int carry = 0; // intialising our carry with 0 intiall
        // while loop will run, until l1 OR l2 not reaches null OR if they both
reaches null. But our carry has some value in it.
        // We will add that as well into our list
        while(l1 != null || l2 != null || carry == 1){
            int sum = 0; // intialising our sum
            if(l1 != null){ // adding l1 to our sum & moving l1
                sum += l1.val;
                l1 = l1.next;
            }
            if(l2 != null){ // adding l2 to our sum & moving l2
                sum += l2.val;
                l2 = l2.next;
            }
            sum += carry; // if we have carry then add it into our sum
            carry = sum/10; // if we get carry, then divide it by 10 to get the
carry
            ListNode node = new ListNode(sum % 10); // the value we'll get by
moduloing it, will become as new node so. add it to our list
            curr.next = node; // curr will point to that new node if we get
            curr = curr.next; // update the current every time
        }
        return dummy.next; // return dummy.next bcz, we don't want the value we
have consider in it intially!!
    }


}
```

Delete all occurrences of a key in DLL

05 October 2024 11:00

```
Solution {
  public static Node deleteAllOccurrences(Node head, int k)
  {
     Node curr=head;
     while(curr!=null)
     {
        if(curr.data==k)
        {
           Node nextnode=curr.next;
           if(curr.prev!=null)//beech ka node ki value k ke barabar hai
           {
              curr.prev.next=curr.next;
           }
           else
           {
              head=curr.next;
           }

           if(curr.next!=null)//beech ka node ki value k ke barabar hai
           {
              curr.next.prev=curr.prev;
           }
           curr=nextnode;
        }
        else
        {
           curr=curr.next;
        }
     }
     return head;
  }
```

Find pairs with given sum in DLL

05 October 2024 11:00

```
class Node {
   int data;
   Node next, prev;
   Node(int data) {
      this.data = data;
   }
}

public class DLLPairSum {

   public void findPairsWithSum(Node head, int target) {
      if (head == null) return;

      Node start = head;
      Node end = head;

      // Move end to the tail
      while (end.next != null) {
         end = end.next;
      }

      boolean found = false;

      // Two-pointer approach
      while (start != null && end != null && start != end && end.next != start) {
         int sum = start.data + end.data;

         if (sum == target) {
            System.out.println("(" + start.data + ", " + end.data + ")");
            found = true;
            start = start.next;
            end = end.prev;
         } else if (sum < target) {
            start = start.next;
         } else {
            end = end.prev;
         }
      }

      if (!found) {
         System.out.println("No pairs found.");
      }
   }
}
```

Remove duplicates from sorted DLL

05 October 2024 11:00

```
Node * removeDuplicates(Node *head)
{
    Node* current = head;
        while (current != NULL && current->next != NULL) {
            if (current->data == current->next->data) {
                Node* duplicate = current->next;
                // Skip the duplicate node
                current->next = duplicate->next;
                // Fix the prev pointer of the next node, if it's not null
                if (duplicate->next != NULL) {
                    duplicate->next->prev = current;
                }
                free(duplicate);
            } else {
                current = current->next;
            }
        }
        return head;
}
```

Reverse LL in group of given size K****

05 October 2024 11:00

```
private ListNode reverse(ListNode head) {
    ListNode prev = null, curr = head;
    while (curr != null) {
        ListNode nextNode = curr.next;
        curr.next = prev;
        prev = curr;
        curr = nextNode;
    }
    return prev;
}

// Find the kth node from current node
private ListNode findKth(ListNode head, int k) {
    k--;
    while (head != null && k > 0) {
        head = head.next;
        k--;
    }
    return head;
}

public ListNode reverseKGroup(ListNode head, int k) {
    ListNode dummy = new ListNode(-1);
    dummy.next = head;

    ListNode groupPrev = dummy;
    ListNode temp = head;

    while (temp != null) {
        ListNode kth = findKth(temp, k);
        if (kth == null) {
            break;
        }

        ListNode groupNext = kth.next;
        kth.next = null;

        // Reverse this k-group
        ListNode newGroupHead = reverse(temp);

        // Link previous part to the new head
        groupPrev.next = newGroupHead;

        // Link end of reversed group to the next part
        temp.next = groupNext;

        // Move pointers forward
        groupPrev = temp;
        temp = groupNext;
    }

    return dummy.next;
}
```

Rotate a LL

05 October 2024 11:00

```
public ListNode rotateRight(ListNode head, int k)
    {
        if (head == null || head.next == null || k == 0) return head;
        ListNode fast=head;
        ListNode slow=head;
        int cnt=1;
        while(fast.next!=null)
        {
            fast=fast.next;
            cnt++;
        }
        k=k%cnt;
        if(k==0)
        {
            return head;
        }
        fast=head;
        while(fast!=null&&k>0)
        {
            fast=fast.next;
            k--;
        }
        if(fast==null)
        {
            return head;
        }
        while(fast.next!=null)
        {
            slow=slow.next;
            fast=fast.next;
        }
        ListNode temp=slow.next;
        slow.next=null;
        fast.next=head;
        head=temp;
        return head;

    }
```

Flattening of LL******

05 October 2024 11:01

Flattening of LL

From <https://takeuforward.org/strivers-a2z-dsa-course/strivers-a2z-dsa-course-sheet-2/>

Flattening of LL

From <https://takeuforward.org/strivers-a2z-dsa-course/strivers-a2z-dsa-course-sheet-2/>

Clone a Linked List with random and next pointer*****

05 October 2024 11:01