**Assignment Questions 12**

**Question 1**

Given a singly linked list, delete **middle** of the linked list. For example, if given linked list is 1->2->**3**->4->5 then linked list should be modified to 1->2->4->5.If there are **even** nodes, then there would be **two middle** nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.If the input linked list is NULL or has 1 node, then it should return NULL

**Example 1:**

Input:

LinkedList: 1->2->3->4->5

Output:1 2 4 5

**Example 2:**

Input:

LinkedList: 2->4->6->7->5->1

Output:2 4 6 5 1

Solve:-

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNodedeleteMiddle(ListNode head) {

if (head == null || head.next == null) {

return null;

}

ListNodeslowPointer = head;

ListNodefastPointer = head;

ListNode previous = null;

while (fastPointer != null &&fastPointer.next != null) {

previous = slowPointer;

slowPointer = slowPointer.next;

fastPointer = fastPointer.next.next;

}

previous.next = slowPointer.next;

return head;

}

}

**Question 2**

Given a linked list of **N** nodes. The task is to check if the linked list has a loop. Linked list can contain self loop.

**Example 1:**

Input:

N = 3

value[] = {1,3,4}

x(position at which tail is connected) = 2

Output:True

Explanation:In above test case N = 3.

The linked list with nodes N = 3 is

given. Then value of x=2 is given which

means last node is connected with xth

node of linked list. Therefore, there

exists a loop.

**Example 2:**

Input:

N = 4

value[] = {1,8,3,4}

x = 0

Output:False

Explanation:For N = 4 ,x = 0 means

then lastNode->next = NULL, then

the Linked list does not contains

any loop.

Solve:-

public class Solution {

public booleanhasCycle(ListNode head) {

ListNode slow=head, fast=head;

while(fast!=null &&fast.next!=null){

fast=fast.next.next;

slow=slow.next;

if(fast==slow) return true;

}

return false;

}

}

**Question 3**

Given a linked list consisting of **L** nodes and given a number **N**. The task is to find the **N**th node from the end of the linked list.

**Example 1:**

Input:

N = 2

LinkedList: 1->2->3->4->5->6->7->8->9

Output:8

Explanation:In the first example, there

are 9 nodes in linked list and we need

to find 2nd node from end. 2nd node

from end is 8.

Example 2

Input:

N = 5

LinkedList: 10->5->100->5

Output:-1

Explanation:In the second example, there

are 4 nodes in the linked list and we

need to find 5th from the end. Since 'n'

is more than the number of nodes in the

linked list, the output is -1.

Solve

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNoderemoveNthFromEnd(ListNode head, int n) {

ListNode start = new ListNode();

start.next = head;

ListNode fast = start;

ListNode slow = start;

for(int i = 1; i<= n; ++i)

fast = fast.next;

while(fast.next != null)

{

fast = fast.next;

slow = slow.next;

}

slow.next = slow.next.next;

return start.next;

}

}

**Question 4**

Given a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.

**Examples:**

Input: R->A->D->A->R->NULL

**Output:** Yes

**Input:** C->O->D->E->NULL

**Output:** No

Solve:-

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public booleanisPalindrome(ListNode head) {

if(head == null || head.next == null){

return true;

}

ListNode slow = head;

ListNode fast = head;

while(fast != null &&fast.next != null){

slow = slow.next;

fast = fast.next.next;

}

ListNodecurr = slow;

ListNodeprev = null;

ListNode next;

while(curr != null){

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

ListNode head1 = head;

ListNode head2 = prev;

while(head2 != null && head1 != null){

if(head1.val != head2.val){

return false;

}

head1 = head1.next;

head2 = head2.next;

}

return true;

}

}

**Question 5**

Given a linked list of **N** nodes such that it may contain a loop.

A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.

Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.

**Example 1**

Input:

N = 3

value[] = {1,3,4}

X = 2

Output:1

Explanation:The link list looks like

1 -> 3 -> 4

^ |

|\_\_\_\_|

A loop is present. If you remove it

successfully, the answer will be 1.

**Example 2:**

Input:

N = 4

value[] = {1,8,3,4}

X = 0

Output:1

Explanation:The Linked list does not

contains any loop.

**Solve :-**

import java.util.HashSet;

import java.util.Set;

// A Linked List Node

class Node

{

int data;

Node next;

}

class Main

{

// Utility function to create a new node with the given data and

// pushes it onto the list's front

public static Node push(Node head, int data)

{

Node node = new Node();

node.data = data;

node.next = head;

return node;

}

// Utility function to print a linked list

public static void printList(Node head)

{

Node curr = head;

while (curr != null)

{

System.out.print(curr.data + " —> ");

curr = curr.next;

}

System.out.println("null");

}

// Function to identify and remove cycle in a linked list using hashing

public static void removeCycle(Node head)

{

Node prev = null; // previous pointer

Node curr = head; // main pointer

// maintain a set to store visited nodes

Set<Node> set = new HashSet<>();

// traverse the list

while (curr != null)

{

// set the previous pointer to null if the current node is seen before

if (set.contains(curr))

{

prev.next = null;

return;

}

// insert the current node into the set

set.add(curr);

// update the previous pointer to the current node and

// move the main pointer to the next node

prev = curr;

curr = curr.next;

}

}

public static void main(String[] args)

{

// total number of nodes in the linked list

int n = 5;

// construct a linked list

Node head = null;

for (int i = n; i> 0; i--) {

head = push(head, i);

}

// insert cycle

head.next.next.next.next.next = head.next;

removeCycle(head);

printList(head);

}

}

**Question 6**

Given a linked list and two integers M and N. Traverse the linked list such that you retain M nodes then delete next N nodes, continue the same till end of the linked list.

Difficulty Level: Rookie

**Examples**:

Input:

M = 2, N = 2

Linked List: 1->2->3->4->5->6->7->8

Output:

Linked List: 1->2->5->6

Input:

M = 3, N = 2

Linked List: 1->2->3->4->5->6->7->8->9->10

Output:

Linked List: 1->2->3->6->7->8

Input:

**M = 1, N = 1**

**Linked List: 1->2->3->4->5->6->7->8->9->10**

**Output:**

**Linked List: 1->3->5->7->9**

Solve:-

import java.util.\*;

class GFG

{

// A linked list node

static class Node

{

int data;

Node next;

};

/\* Function to insert a node at the beginning \*/

static Node push( Node head\_ref, int new\_data)

{

Node new\_node = new Node();

/\* put in the data \*/

new\_node.data = new\_data;

/\* link the old list of the new node \*/

new\_node.next = (head\_ref);

/\* move the head to point to the new node \*/

(head\_ref) = new\_node;

return head\_ref;

}

static void printList( Node head)

{

Node temp = head;

while (temp != null)

{

System.out.printf("%d ", temp.data);

temp = temp.next;

}

System.out.printf("\n");

}

// Function to skip M nodes and then

// delete N nodes of the linked list.

static void skipMdeleteN( Node head, int M, int N)

{

Node curr = head, t;

int count;

// The main loop that traverses

while (curr!=null)

{

// Skip M nodes

for (count = 1; count < M &&curr != null; count++)

curr = curr.next;

// If we reached end of list, then return

if (curr == null)

return;

// Start from next node and delete N nodes

t = curr.next;

for (count = 1; count <= N && t != null; count++)

{

Node temp = t;

t = t.next;

}

// Link the previous list with remaining nodes

curr.next = t;

// Set current pointer for next iteration

curr = t;

}

}

public static void main(String args[])

{

/\* Create following linked list

1.2.3.4.5.6.7.8.9.10 \*/

Node head = null;

int M=2, N=3;

head=push(head, 10);

head=push(head, 9);

head=push(head, 8);

head=push(head, 7);

head=push(head, 6);

head=push(head, 5);

head=push(head, 4);

head=push(head, 3);

head=push(head, 2);

head=push(head, 1);

System.out.printf("M = %d, N = %d \nGiven" +

"Linked list is :\n", M, N);

printList(head);

skipMdeleteN(head, M, N);

System.out.printf("\nLinked list after deletion is :\n");

printList(head);

}

}

**Question 7**

Given two linked lists, insert nodes of second list into first list at alternate positions of first list. For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Use of extra space is not allowed (Not allowed to create additional nodes), i.e., insertion must be done in-place. Expected time complexity is O(n) where n is number of nodes in first list.

Solve:-

class LinkedList

{

Node head; // head of list

/\* Linked list Node\*/

class Node

{

int data;

Node next;

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

}

/\* Inserts a new Node at front of the list. \*/

void push(int new\_data)

{

/\* 1 & 2: Allocate the Node &

Put in the data\*/

Node new\_node = new Node(new\_data);

/\* 3. Make next of new Node as head \*/

new\_node.next = head;

/\* 4. Move the head to point to new Node \*/

head = new\_node;

}

// Main function that inserts nodes of linked list q into p at

// alternate positions. Since head of first list never changes

void merge(LinkedList q)

{

Node p\_curr = head, q\_curr = q.head;

Node p\_next, q\_next;

// While there are available positions in p;

while (p\_curr != null &&q\_curr != null) {

// Save next pointers

p\_next = p\_curr.next;

q\_next = q\_curr.next;

// make q\_curr as next of p\_curr

q\_curr.next = p\_next; // change next pointer of q\_curr

p\_curr.next = q\_curr; // change next pointer of p\_curr

// update current pointers for next iteration

p\_curr = p\_next;

q\_curr = q\_next;

}

q.head = q\_curr;

}

/\* Function to print linked list \*/

void printList()

{

Node temp = head;

while (temp != null)

{

System.out.print(temp.data+" ");

temp = temp.next;

}

System.out.println();

}

/\* Driver program to test above functions \*/

public static void main(String args[])

{

LinkedList llist1 = new LinkedList();

LinkedList llist2 = new LinkedList();

llist1.push(3);

llist1.push(2);

llist1.push(1);

System.out.println("First Linked List:");

llist1.printList();

llist2.push(8);

llist2.push(7);

llist2.push(6);

llist2.push(5);

llist2.push(4);

System.out.println("Second Linked List:");

llist1.merge(llist2);

System.out.println("Modified first linked list:");

llist1.printList();

System.out.println("Modified second linked list:");

llist2.printList();

}

}

**Question 8**

Given a singly linked list, find if the linked list is [circular](https://www.geeksforgeeks.org/circular-linked-list/amp/) or not.

A linked list is called circular if it is not NULL-terminated and all nodes are connected in the form of a cycle. Below is an example of a circular linked list.

Solve:-

import java.util.\*;

class GFG {

/\* Link list Node \*/

static class Node {

int data;

Node next;

}

/\*This function returns true if given linked

list is circular, else false. \*/

static booleanisCircular(Node head)

{

if (head == null)

return true;

Node node = head.next;

while (node != null && node != head)

node = node.next;

return (node == head);

}

static Node newNode(int data)

{

Node temp = new Node();

temp.data = data;

temp.next = null;

return temp;

}

public static void main(String args[])

{

Node head = newNode(1);

head.next = newNode(2);

head.next.next = newNode(3);

head.next.next.next = newNode(4);

System.out.print(isCircular(head) ? "Yes\n"

: "No\n");

head.next.next.next.next = head;

System.out.print(isCircular(head) ? "Yes\n"

: "No\n");

}

}