GeeksforGeeks
A computer science portal for geeks

Practice IDE Q&A GeeksQuiz

Rearrange a given linked list in-place.

Given a singly linked list L_0 -> L_1 -> … -> L_{n-1} -> L_n. Rearrange the nodes in the list so that the new formed list is : L_0 -> L_n -> L_1 -> L_{n-1} -> L_2 -> L_{n-2} …

You are required do this in-place without altering the nodes' values.

```
Examples:
Input:  1 -> 2 -> 3 -> 4
Output: 1 -> 4 -> 2 -> 3

Input:  1 -> 2 -> 3 -> 4 -> 5
Output: 1 -> 5 -> 2 -> 4 -> 3
```

We strongly recommend you to minimize your browser and try this yourself first.

Simple Solution

```
1) Initialize current node as head.
2) While next of current node is not null, do following
    a) Find the last node, remove it from end and insert it as next
       of current node.
    b) Move current to next to next of current
```

Time complexity of the above simple solution is $O(n^2)$ where n is number of nodes in linked list.

Efficient Solution:

```
1) Find the middle point using tortoise and hare method.
2) Split the linked list in two halves using found middle point in step 1.
3) Reverse the second half.
4) Do alternate merge of first and second halves.
```

Time Complexity of this solution is O(n).

Below is the implementation of this method.

C++

```
// C++ program to rearrange a linked list in-place
#include<bits/stdc++.h>
using namespace std;

// Linkedlist Node structure
struct Node
{
    int data;
    struct Node *next;
};

// Function to create newNode in a linkedlist
Node* newNode(int key)
{
    Node *temp = new Node;
    temp->data = key;
    temp->next = NULL;
    return temp;
}

// Function to reverse the linked list
void reverselist(Node **head)
{
    // Initialize prev and current pointers
    Node *prev = NULL, *curr = *head, *next;

    while (curr)
    {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    *head = prev;
}

// Function to print the linked list
void printlist(Node *head)
{
    while (head != NULL)
    {
        cout << head->data << " ";
        if(head->next) cout << "-> ";
        head = head->next;
    }
    cout << endl;
}

// Function to rearrange a linked list
void rearrange(Node **head)
{
    // 1) Find the muddle point using tortoise and hare method
    Node *slow = *head, *fast = slow->next;
    while (fast && fast->next)
    {
        slow = slow->next;
        fast = fast->next->next;
    }

    // 2) Split the linked list in two halves
    // head1, head of first half    1 -> 2
    // head2, head of second half   3 -> 4
    Node *head1 = *head;
    Node *head2 = slow->next;
    slow->next = NULL;

    // 3) Reverse the second half, i.e.,  4 -> 3
    reverselist(&head2);

    // 4) Merge alternate nodes
    *head = newNode(0); // Assign dummy Node

    // curr is the pointer to this dummy Node, which will
    // be used to form the new list
    Node *curr = *head;
    while (head1 || head2)
    {
        // First add the element from list
        if (head1)
        {
            curr->next = head1;
            curr = curr->next;
            head1 = head1->next;
        }

        // Then add the element from second list
        if (head2)
        {
            curr->next = head2;
            curr = curr->next;
            head2 = head2->next;
        }
    }

    // Assign the head of the new list to head pointer
    *head = (*head)->next;
}

// Driver program
int main()
{
    Node *head = newNode(1);
    head->next = newNode(2);
    head->next->next = newNode(3);
    head->next->next->next = newNode(4);
    head->next->next->next->next = newNode(5);

    printlist(head);    // Print original list
    rearrange(&head);       // Modify the list
    printlist(head);    // Print modified list
    return 0;
}
```

Run on IDE

Java

```
// Java program to rearrange link list in place

// Linked List Class
class LinkedList {
    static Node head;  // head of list


    /* Node Class */
    static class Node {

        int data;
        Node next;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = null;
        }
    }

    void printlist(Node node) {
        if (node == null) {
            return;
        }
        while (node != null) {
            System.out.print(node.data + " -> ");
            node = node.next;
        }
    }

    Node reverselist(Node node) {
        Node prev = null, curr = node, next;
        while (curr != null) {
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        node = prev;
        return node;
    }

    void rearrange(Node node) {

        // 1) Find the middle point using tortoise and hare method
        Node slow = node, fast = slow.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        // 2) Split the linked list in two halves
        // node1, head of first half    1 -> 2 -> 3
        // node2, head of second half   4 -> 5
        Node node1 = node;
        Node node2 = slow.next;
        slow.next = null;

        // 3) Reverse the second half, i.e., 5 -> 4
        node2 = reverselist(node2);

        // 4) Merge alternate nodes
        node = new Node(0); // Assign dummy Node

        // curr is the pointer to this dummy Node, which will
        // be used to form the new list
        Node curr = node;
        while (node1 != null || node2 != null) {
            // First add the element from first list
            if (node1 != null) {
                curr.next = node1;
                curr = curr.next;
                node1 = node1.next;
            }

            // Then add the element from second list
            if (node2 != null) {
                curr.next = node2;
                curr = curr.next;
                node2 = node2.next;
            }
        }

        // Assign the head of the new list to head pointer
        node = node.next;
    }

    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);

        list.printlist(head); // print original list
        list.rearrange(head); // rearrange list as per ques
        System.out.println("");
        list.printlist(head); // print modified list

    }
}

// This code has been contributed by Mayank Jaiswal
```

Run on IDE

Output:

```
1 -> 2 -> 3 -> 4 -> 5
1 -> 5 -> 2 -> 4 -> 3
```

Thanks to Gaurav Ahirwar for suggesting above approach.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

19 Comments Category: Linked Lists

Related Posts:

- Merge two sorted linked lists such that merged list is in reverse order
- Compare two strings represented as linked lists
- Sort a linked list that is sorted alternating ascending and descending orders?
- Select a Random Node from a Singly Linked List
- Merge Sort for Doubly Linked List
- Point to next higher value node in a linked list with an arbitrary pointer
- Swap nodes in a linked list without swapping data
- Generic Linked List in C

(Login to Rate and Mark)

3.4 Average Difficulty : **3.4/5.0**
Based on **7** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 20 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

@geeksforgeeks, Some rights reserved Contact Us! About Us! Advertise with us!