

GeeksforGeeks

A computer science portal for geeks

Practice

IDE

Q&A

GeeksQuiz

Flattening a Linked List

Given a linked list where every node represents a linked list and contains two pointers of its type:

- (i) Pointer to next node in the main list (we call it 'right' pointer in below code)
- (ii) Pointer to a linked list where this node is head (we call it 'down' pointer in below code).

All linked lists are sorted. See the following example

```

5 -> 10 -> 19 -> 28
|   |   |   |
v   v   v   v
7   20  22  35
|       |   |
v       v   v
8       50  40
|       |   |
v       v   v
30      45

```

Write a function `flatten()` to flatten the lists into a single linked list. The flattened linked list should also be sorted. For example, for the above input list, output list should be 5->7->8->10->19->20->22->28->30->35->40->45->50.

The idea is to use Merge() process of [merge sort for linked lists](#). We use `merge()` to merge lists one by one. We recursively merge() the current list with already flattened list.

The down pointer is used to link nodes of the flattened list.

Following are C and Java implementations.

C/C++

```

// C program for flattening a linked list
#include <stdio.h>
#include <stdlib.h>

// A Linked List Node
typedef struct Node
{
    int data;

```

```
    struct Node *right;
    struct Node *down;
} Node;

/* A utility function to insert a new node at the beginning
of linked list */
void push (Node** head_ref, int new_data)
{
    /* allocate node */
    Node* new_node = (Node *) malloc(sizeof(Node));
    new_node->right = NULL;

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->down = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Function to print nodes in the flattened linked list */
void printList(Node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->down;
    }
}

// A utility function to merge two sorted linked lists
Node* merge( Node* a, Node* b )
{
    // If first list is empty, the second list is result
    if (a == NULL)
        return b;

    // If second list is empty, the second list is result
    if (b == NULL)
        return a;

    // Compare the data members of head nodes of both lists
    // and put the smaller one in result
    Node* result;
    if (a->data < b->data)
    {
        result = a;
        result->down = merge( a->down, b );
    }
}
```

```

    }
    else
    {
        result = b;
        result->down = merge( a, b->down );
    }

    return result;
}

// The main function that flattens a given linked list
Node* flatten (Node* root)
{
    // Base cases
    if (root == NULL || root->right == NULL)
        return root;

    // Merge this list with the list on right side
    return merge( root, flatten(root->right) );
}

// Driver program to test above functions
int main()
{
    Node* root = NULL;

    /* Let us create the following linked list
    5 -> 10 -> 19 -> 28
    |   |   |   |
    V   V   V   V
    7   20  22  35
    |       |   |
    V       V   V
    8       50  40
    |       |   |
    V       V   V
    30      45

    */
    push( &root, 30 );
    push( &root, 8 );
    push( &root, 7 );
    push( &root, 5 );

    push( &( root->right ), 20 );
    push( &( root->right ), 10 );

    push( &( root->right->right ), 50 );
    push( &( root->right->right ), 22 );
    push( &( root->right->right ), 19 );

```

```

push( &(amp; root->right->right->right ), 45 );
push( &(amp; root->right->right->right ), 40 );
push( &(amp; root->right->right->right ), 35 );
push( &(amp; root->right->right->right ), 20 );

// Let us flatten the list
root = flatten(root);

// Let us print the flattened linked list
printList(root);

return 0;
}

```

Java

```

// Java program for flattening a Linked List
class LinkedList
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node right, down;
        Node(int data)
        {
            this.data = data;
            right = null;
            down = null;
        }
    }

    // An utility function to merge two sorted linked lists
    Node merge(Node a, Node b)
    {
        // if first linked list is empty then second
        // is the answer
        if (a == null) return b;

        // if second linked list is empty then first
        // is the result
        if (b == null) return a;

        // compare the data members of the two linked lists
        // and put the larger one in the result
        Node result;
    }
}

```

```
    if (a.data < b.data)
    {
        result = a;
        result.down = merge(a.down, b);
    }

    else
    {
        result = b;
        result.down = merge(a, b.down);
    }

    return result;
}

Node flatten(Node root)
{
    // Base Cases
    if (root == null || root.right == null)
        return root;

    // recur for list on right
    root.right = flatten(root.right);

    // now merge
    root = merge(root, root.right);

    // return the root
    // it will be in turn merged with its left
    return root;
}

/* Utility function to insert a node at beginning of the
   linked list */
Node push(Node head_ref, int data)
{
    /* 1 & 2: Allocate the Node &
       Put in the data*/
    Node new_node = new Node(data);

    /* 3. Make next of new Node as head */
    new_node.down = head_ref;

    /* 4. Move the head to point to new Node */
    head_ref = new_node;

    /*5. return to link it back */
    return head_ref;
}
```

```

void printList()
{
    Node temp = head;
    while (temp != null)
    {
        System.out.print(temp.data + " ");
        temp = temp.down;
    }
    System.out.println();
}

/* Drier program to test above functions */
public static void main(String args[])
{
    LinkedList L = new LinkedList();

    /* Let us create the following linked list
        5 -> 10 -> 19 -> 28
        |   |   |   |
        V   V   V   V
        7   20  22  35
        |       |   |
        V       V   V
        8       50  40
        |           |
        V           V
        30          45
    */

    L.head = L.push(L.head, 30);
    L.head = L.push(L.head, 8);
    L.head = L.push(L.head, 7);
    L.head = L.push(L.head, 5);

    L.head.right = L.push(L.head.right, 20);
    L.head.right = L.push(L.head.right, 10);

    L.head.right.right = L.push(L.head.right.right, 50);
    L.head.right.right = L.push(L.head.right.right, 22);
    L.head.right.right = L.push(L.head.right.right, 19);

    L.head.right.right.right = L.push(L.head.right.right.right, 45);
    L.head.right.right.right = L.push(L.head.right.right.right, 40);
    L.head.right.right.right = L.push(L.head.right.right.right, 35);
    L.head.right.right.right = L.push(L.head.right.right.right, 20);

    // flatten the list
    L.head = L.flatten(L.head);

    L.printList();
}

```

```
}  
} /* This code is contributed by Rajat Mishra */
```

Output:

5 7 8 10 19 20 20 22 30 35 40 45 50

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



68 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](#)
- [Rearrange a given linked list in-place.](#)
- [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](#)

([Login](#) to Rate and Mark)

3.6

Average Difficulty : **3.6/5.0**
Based on **3** vote(s)

☐
☐

Add to TODO List

Mark as DONE

Like Share 7 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!](#)