Segregate Even and Odd Nodes

Given a Linked List of integers, write a function to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list. Also, keep the order of even and odd numbers the same.

Examples:

```
Input: 17->15->8->12->10->5->4->1->7->6->NULL
Output: 8->12->10->4->6->17->15->5->1->7->NULL

Input: 8->12->10->5->4->1->6->NULL
Output: 8->12->10->4->6->5->1->NULL

// If all numbers are even then do not change the list
Input: 8->12->10->NULL
Output: 8->12->10->NULL

// If all numbers are odd then do not change the list
Input: 1->3->5->7->NULL
Output: 1->3->5->7->NULL
```

The idea is to split the linked list into two: one containing all even nodes and the other containing all odd nodes. And finally, attach the odd node linked list after the even node linked list.
To split the Linked List, traverse the original Linked List and move all odd nodes to a separate Linked List of all odd nodes. At the end of loop, the original list will have all the even nodes and the odd node list will have all the odd

nodes. To keep the ordering of all nodes same, we must insert all the odd nodes at the end of the even node list. And to do that in constant time, we must keep track of last pointer in the even node list.

Below is the implementation of the above approach:

C++ Java

```
// CPP program to segregate even and odd nodes in a
// Linked List
#include <bits/stdc++.h>
using namespace std;

/* a node of the singly linked list */
struct Node {
    int data;
    struct Node* next;
};

// Function to segregate even and odd nodes.
void segregateEvenOdd(struct Node** head_ref)
{
    // Starting node of list having even values.
    Node* evenStart = NULL;
    // Ending node of even values list.
    Node* evenEnd = NULL;
    // Starting node of odd values list.
    Node* oddStart = NULL;
    // Ending node of odd values list.
    Node* oddEnd = NULL;
    // Node to traverse the list.
```

```
Node* currNode = *head_ref;

while (currNode != NULL) {
    int val = currNode->data;

    // If current value is even, add it to even values
    // list.
    if (val % 2 == 0) {
        if (evenStart == NULL) {
            evenStart = currNode;
            evenEnd = evenStart;
        }
        else {
            evenEnd->next = currNode;
            evenEnd = evenEnd->next;
        }
    }

    // If current value is odd, add it to odd values
    // list.
    else {
        if (oddStart == NULL) {
            oddStart = currNode;
            oddEnd = oddStart;
        }
        else {
            oddEnd->next = currNode;
            oddEnd = oddEnd->next;
        }
```

```
        }

        // Move head pointer one step in forward direction
        currNode = currNode->next;
    }

    // If either odd list or even list is empty, no change
    // is required as all elements are either even or odd.
    if (oddStart == NULL || evenStart == NULL)
        return;

    // Add odd list after even list.
    evenEnd->next = oddStart;
    oddEnd->next = NULL;

    // Modify head pointer to starting of even list.
    *head_ref = evenStart;
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginning */
void push(Node** head_ref, int new_data)
{
    /* allocate node */
    Node* new_node = new Node();
    /* put in the data */
    new_node->data = new_data;
    /* link the old list off the new node */
    new_node->next = (*head_ref);
```

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

/* Function to print nodes in a given linked list */
void printList(Node* node)
{
    while (node != NULL) {
        cout << node->data << " ";
        node = node->next;
    }
}

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty list */
    Node* head = NULL;

    /* Let us create a sample linked list as following
    0->1->4->6->9->10->11 */

    push(&head, 11);
    push(&head, 10);
    push(&head, 9);
    push(&head, 6);
    push(&head, 4);
```

```
    cout << "Original Linked list" << endl;
    printList(head);

    segregateEvenOdd(&head);

    cout << endl << "Modified Linked list " << endl;
    printList(head);

    return 0;
}
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

Output

```
Original Linked list
0 1 4 6 9 10 11
Modified Linked list
0 4 6 10 1 9 11
```

Time Complexity: O(n), As we are only traversing linearly through the list.
Auxiliary Space: O(1)

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.