Remove duplicates from a sorted linked list

Write a removeDuplicates() function which takes a list sorted in non-decreasing order and deletes any duplicate nodes from the list. The list should only be traversed once.

For example if the linked list is 11->11->11->21->43->43->60 then removeDuplicates() should convert the list to 11->21->43->60.

Algorithm:

Traverse the list from the head (or start) node. While traversing, compare each node with its next node. If data of next node is same as current node then delete the next node. Before we delete a node, we need to store next pointer of the node

Implementation:

Functions other than removeDuplicates() are just to create a linked linked list and test removeDuplicates().

C

```
/* C Program to remove duplicates from a sorted linked list */
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* The function removes duplicates from a sorted list */
void removeDuplicates(struct node* head)
{
    /* Pointer to traverse the linked list */
    struct node* current = head;

    /* Pointer to store the next pointer of a node to be deleted*/
    struct node* next_next;

    /* do nothing if the list is empty */
    if (current == NULL)
       return;

    /* Traverse the list till last node */
    while (current->next != NULL)
    {
       /* Compare current node with next node */
       if (current->data == current->next->data)
       {
           /* The sequence of steps is important*/
           next_next = current->next->next;
           free(current->next);
           current->next = next_next;
       }
       else /* This is tricky: only advance if no deletion */
       {
          current = current->next;
       }
    }
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginging of the linked list */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
```

```
    struct node* new_node =
            (struct node*) malloc(sizeof(struct 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(struct node *node)
{
    while (node!=NULL)
    {
       printf("%d ", node->data);
       node = node->next;
    }
}

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

    /* Let us create a sorted linked list to test the functions
     Created linked list will be 11->11->11->13->13->20 */
    push(&head, 20);
    push(&head, 13);
    push(&head, 13);
    push(&head, 11);
    push(&head, 11);
    push(&head, 11);

    printf("\n Linked list before duplicate removal  ");
    printList(head);

    /* Remove duplicates from linked list */
    removeDuplicates(head);

    printf("\n Linked list after duplicate removal ");
    printList(head);

    return 0;
}
```

Java

```
// Java program to remove duplicates from a sorted linked list
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    void removeDuplicates()
    {
        /*Another reference to head*/
        Node current = head;
```

```
        /* Pointer to store the next pointer of a node to be deleted*/
        Node next_next;

        /* do nothing if the list is empty */
        if (head == null)
            return;

        /* Traverse list till the last node */
        while (current.next != null) {

            /*Compare current node with the next node */
            if (current.data == current.next.data) {
                next_next = current.next.next;
                current.next = null;
                current.next = next_next;
            }
            else // advance if no deletion
               current = current.next;
        }
    }

    /* Utility functions */

    /* Inserts a new Node at front of the list. */
    public 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;
    }

    /* Function to print linked list */
     void printList()
     {
         Node temp = head;
         while (temp != null)
         {
            System.out.print(temp.data+" ");
            temp = temp.next;
         }
         System.out.println();
     }

     /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();
        llist.push(20);
        llist.push(13);
        llist.push(13);
        llist.push(11);
        llist.push(11);
        llist.push(11);

        System.out.println("List before removal of duplicates");
        llist.printList();

        llist.removeDuplicates();

        System.out.println("List after removal of elements");
        llist.printList();
    }
}
/* This code is contributed by Rajat Mishra */
```

Output:

```
Linked list before duplicate removal 11 11 11 13 13 20
Linked list after duplicate removal 11 13 20
```

Time Complexity: O(n) where n is number of nodes in the given linked list.

References:

cslibrary.stanford.edu/105/LinkedListProblems.pdf