

Remove duplicates from an unsorted linked list

Write a removeDuplicates() function which takes a list and deletes any duplicate nodes from the list. The list is not sorted.

For example if the linked list is 12->11->12->21->41->43->21 then removeDuplicates() should convert the list to 12->11->21->41->43.

METHOD 1 (Using two loops)

This is the simple way where two loops are used. Outer loop is used to pick the elements one by one and inner loop compares the picked element with rest of the elements.

Thanks to Gaurav Saxena for his help in writing this code.

C

```
/* Program to remove duplicates in an unsorted array */

#include<stdio.h>
#include<stdlib.h>

/* A linked list node */
struct node
{
    int data;
    struct node *next;
};

/* Function to remove duplicates from a unsorted linked list */
void removeDuplicates(struct node *start)
{
    struct node *ptr1, *ptr2, *dup;
    ptr1 = start;

    /* Pick elements one by one */
    while(ptr1 != NULL && ptr1->next != NULL)
    {
        ptr2 = ptr1;

        /* Compare the picked element with rest of the elements */
        while(ptr2->next != NULL)
        {
            /* If duplicate then delete it */
            if(ptr1->data == ptr2->next->data)
            {
                /* sequence of steps is important here */
                dup = ptr2->next;
                ptr2->next = ptr2->next->next;
                free(dup);
            }
            else /* This is tricky */
            {
                ptr2 = ptr2->next;
            }
        }
        ptr1 = ptr1->next;
    }
}

/* UTILITY FUNCTIONS */
/* Function to push a node */
void push(struct node** head_ref, int new_data);

/* Function to print nodes in a given linked list */
void printList(struct node *node);
```

```

/* Driver program to test above function */
int main()
{
    struct node *start = NULL;

    /* The constructed linked list is:
    10->12->11->11->12->11->10*/
    push(&start, 10);
    push(&start, 11);
    push(&start, 12);
    push(&start, 11);
    push(&start, 11);
    push(&start, 12);
    push(&start, 10);

    printf("\n Linked list before removing duplicates ");
    printList(start);

    removeDuplicates(start);

    printf("\n Linked list after removing duplicates ");
    printList(start);

    getchar();
}

/* Function to push a node */
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;
    }
}

```

Java

```

// Java program to remove duplicates from unsorted linked list

class LinkedList {

    static Node head;

    static class Node {

        int data;
        Node next;

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

```

```

}

/* Function to remove duplicates from a unsorted linked list */
void remove_duplicates() {
    Node ptr1 = null, ptr2 = null, dup = null;
    ptr1 = head;

    /* Pick elements one by one */
    while (ptr1 != null && ptr1.next != null) {
        ptr2 = ptr1;

        /* Compare the picked element with rest of the elements */
        while (ptr2.next != null) {

            /* If duplicate then delete it */
            if (ptr1.data == ptr2.next.data) {

                /* sequence of steps is important here */
                dup = ptr2.next;
                ptr2.next = ptr2.next.next;
                System.gc();
            } else /* This is tricky */ {
                ptr2 = ptr2.next;
            }
        }
        ptr1 = ptr1.next;
    }
}

void printList(Node node) {
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
}

public static void main(String[] args) {
    LinkedList list = new LinkedList();
    list.head = new Node(10);
    list.head.next = new Node(12);
    list.head.next.next = new Node(11);
    list.head.next.next.next = new Node(11);
    list.head.next.next.next.next = new Node(12);
    list.head.next.next.next.next.next = new Node(11);
    list.head.next.next.next.next.next.next = new Node(10);

    System.out.println("Linked List before removing duplicates ");
    list.printList(head);

    list.remove_duplicates();
    System.out.println("");
    System.out.println("Linked List after removing duplicates");
    list.printList(head);
}
}

// This code has been contributed by Mayank Jaiswal

```

Time Complexity: $O(n^2)$

METHOD 2 (Use Sorting)

In general, Merge Sort is the best suited sorting algorithm for sorting linked lists efficiently.

- 1) Sort the elements using Merge Sort. We will soon be writing a post about sorting a linked list. $O(n \log n)$
- 2) Remove duplicates in linear time using the [algorithm for removing duplicates in sorted Linked List](#). $O(n)$

Please note that this method doesn't preserve the original order of elements.

Time Complexity: $O(n \log n)$

METHOD 3 (Use Hashing)

We traverse the link list from head to end. For every newly encountered element, we check whether it is in the hash table: if yes, we

remove it; otherwise we put it in the hash table.

Thanks to bearwang for suggesting this method.

Time Complexity: $O(n)$ on average (assuming that hash table access time is $O(1)$ on average).