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;
}
```

Run on IDE

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 */
```

Run on IDE

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

106 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)

1.6 Average Difficulty : **1.6/5.0**
Based on **5** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 5 people like this.

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

106 Comments GeeksforGeeks 1 Login

Recommend 4 Share Sort by Newest

Join the discussion…

Bhavesh Upadhyay • 12 days ago

```
list* remove_duplicates(list* head){

list* tmp=head,*cur;

while(tmp->next){

while(tmp->next && tmp->data==tmp->next->data){

cur=tmp->next;

tmp->next=cur->next;

free(cur);

}

if(tmp->next){

tmp=tmp->next;

}

}

return head;

}
```

• Reply • Share ›

shivam verma • 2 months ago

Another Simple One

```
if(head == null)

return;

while(current != null) {

while(current.next != null && current.next.data == current.data)

current.next = current.next.next;

current = current.next;

}
```

• Reply • Share ›

Jatin • 2 months ago

Simple version

```
public void removeDuplicates() {
Node node = HEAD;
while(node != null && node.getNext() != null){
if( node.getData() == node.getNext().getData()){
node.setNext(node.getNext().getNext());
}
node = node.getNext();
}
}
```

• Reply • Share ›

Manpreet Lehal • 3 months ago

this code not working properly for the input linked list 11 12 11 12 13 20

• Reply • Share ›

Aayush Mudgal → Manpreet Lehal • 3 months ago

It is not a sorted list, the code is for a sorted list ONLY!

• Reply • Share ›

girlwhoCodes • 3 months ago

why do we use (*head_ref) instead of *head_ref?

• Reply • Share ›

vimal → girlwhoCodes • 3 months ago

in this case you can also use *head_ref no prob with that... but if we do (*head_ref) than it you can easily tell that is de-refrencing and not the multiplication .... well its just that it looks clear what we are doing by using proper brackets..

its like we can do->
if(i>0)
do-somthing
but here we can also do like ->
if(i>0){
do-something
}
.. so it looks clean.. u got it right ?

1 • Reply • Share ›

girlwhoCodes → vimal • 3 months ago

Thanks Vimal!

• Reply • Share ›

Roopa • 4 months ago

how to delete only fibonacci value only out of all the elements in a single linked list??

• Reply • Share ›

Raj Kumar Chauhan • 4 months ago

typo in header stdlib.h

@GeeksforGeeks

• Reply • Share ›

cpuxtech • 4 months ago

It's worth noting that this algorithm, when used in a Singly Linked List class implementation will not update the tail if the tail is also a duplicate value so we will have one memory leak and an invalid tail reference if I am not mistaken...to remedy this I have made the following which should be a fairly solid implementation:

```
void SinglyList::removeDuplicates() {
Singly *tmp = head;
Singly *test = head->next;
while(test != nullptr) {
if (tmp->data == test->data){
if(test == tail) {
delete tail;
tail = tmp;
tail->next = nullptr;
return;
}
tmp->next = test->next;
delete test;
test = tmp->next;
} else {
tmp = test;
test = tmp->next;
}
}
}
```

• Reply • Share ›

Anand • 4 months ago

when do pass struct node **head and struct node *head?

do we pass **head when we are modifying linkedlist?and *head while reading linkedlist?

• Reply • Share ›

Raj Kumar Chauhan ➦ Anand • 4 months ago

**head when there is risk of modification of head pointer and *head for all the operation which do not modify head pointer.

read this one.

http://www.geeksforgeeks.org/h...

• Reply • Share ›

kartik mittal • 5 months ago

a simpler implementation:
while(current !=NULL){
if(current->data==current->next->data)
{ temp=current->next;
current->next=temp->next;
free(temp);
}
else current=current->next;
}

• Reply • Share ›

Aayush ➦ kartik mittal • 3 months ago

void removedplicate(node** head)

{

node* current=(*head);

node* temp=(*head);

if(current==NULL)

{

cout<<"List is empty"<<endl; return;="" }="" while(current-="">next!=NULL)

{

temp=current->next;

if(current->data==temp->data)

{

see more

• Reply • Share ›

Akilan Arasu ↪ kartik mittal • 4 months ago

There is only one problem in this implementation: If there is only one node in the linked list, then the if condition will trigger an error, since you're dereferencing a NULL pointer.

Just check for current->next too, and you're solution will be functional too.

^ | ⌄ • Reply • Share ›

Ishir • 5 months ago

I used recursive way..

```
void removeSortDup(struct node** head){

struct node* list = (*head);

if(!list){

return;

}else{

if(list && list->next){

if(list->data == list->next->data){

struct node* next=list->next->next;

delete(list->next);

list->next=next;
```

see more

^ | ⌄ • Reply • Share ›

Dhrtzzz • 6 months ago

Is my code correct..have i met the edge cases??

```
struct node* removedup(struct node* head)
{
struct node *temp;
struct node *curr;
while(curr !=NULL)
{
if(curr->data==curr->next->data)
{ temp=curr->next;
curr->next=temp->next;
free(temp);
}
```

curr=curr->next;
}
}

• Reply • Share ›

Mysterious Mind ➦ Dhrtzzz • 5 months ago

No,because while(curr !=NULL), if curr->next is NULL,then curr->next->data will be invalid.

• Reply • Share ›

Rashika Singhal ➦ Dhrtzzz • 5 months ago

No tihs is not in these case if we have 3+ consecutive repetitions we cant do it

• Reply • Share ›

Ajitesh Mandal • 6 months ago

http://ideone.com/ZyVWBT

• Reply • Share ›

JavaCoder • 6 months ago

Java code:
http://ideone.com/EiTAqD

• Reply • Share ›

Deepak Sharma • 6 months ago

Here is my c++ implementation..
let me know if there are any bugs..
http://ideone.com/oI5uNX

• Reply • Share ›

coolk • 6 months ago

Java approach. Let me know if I missed any boundary conditions.
http://ideone.com/ekHmJq

• Reply • Share ›

BegWorld • 7 months ago

void removeDuplicates(struct node *head)
{
struct node *curr=head;
struct node *nextnode;
while(curr->next!=NULL)
{
nextnode=curr->next;
if(curr->data==nextnode->data)

{
curr->next=nextnode->next;
free(nextnode);
}
curr=curr->next;
}
}
Is this also correct?

Reply • Share ›

Akilan Arasu BegWorld • 4 months ago

For the input 11->11->11->13->13->20:
Your solution will delete the 2nd 11 and the 2nd 13 correctly but not the 3rd 11.
This can be rectified by adding else before curr = curr->next;

Reply • Share ›

surbhijain93 BegWorld • 6 months ago

you need to add else before curr=curr->next;

1 • Reply • Share ›

Astha Banthia BegWorld • 6 months ago

Your code won't work when LL is of the form 1->1->NULL
After the completion of 1st iteration value of current will be NULL and thus accessing current->next will give error.

1 • Reply • Share ›

Mysterious Mind Astha Banthia • 5 months ago

after 1st iteration curr will still point to the head.

Reply • Share ›

Akilan Arasu Mysterious Mind • 4 months ago

No it won't. Astha's right, curr will be NULL after 1st iteration. And you can't dereference a NULL pointer using curr->next in the while condition.

Reply • Share ›

Mysterious Mind Akilan Arasu • 4 months ago

My bad....actually I was referring to GFG's solution.

1 • Reply • Share ›

Gautham Kumaran • 7 months ago

simple java code

https://github.com/gautham20/g...

• Reply • Share ›

sha2832 • 7 months ago

My code to delete duplicate contents from the Linked-List

```
#include<iostream>
#include<conio.h>
using namespace std;
struct node
{
int data;
node *link;
}*head=NULL,*ptr,*temp,*pptr;

void create()
{
int item;
cout<<"enter the item you want to insert"<<endl; cin="">>item;
ptr=head;
if (ptr==NULL)
{
```

see more

• Reply • Share ›

karan • 7 months ago

here is my code:

```
struct node *rem(struct node *head)
{
struct node *p=head,*q=head->next,*r=NULL;
struct node *temp=NULL;

while(p!=NULL && q!=NULL)
{
if( p->data == q->data )
{
r=q;
q=q->next;
}
else
{
p->next=q;
p=p->next;
```

q=q->next;
}

}

return head;
}

^ | ⌄ • Reply • Share ›

santhosh • 7 months ago

awesome code..
especially incrementing pointer only after deleting all the duplicates

2 ^ | ⌄ • Reply • Share ›

Jitin maherchandani • 7 months ago

Another method:-

void del_dup(struct node * head)

{

if (head==NULL)

{

printf("List is empty\n");

return;

}

else

{

struct node *temp1,*temp2;

see more

^ | ⌄ • Reply • Share ›

Akilan Arasu ➦ Jitin maherchandani • 4 months ago

Looks complicated with all the flag variables.

^ | ⌄ • Reply • Share ›

Anand Barnwal ➦ Jitin maherchandani • 7 months ago

I think your solution won't work.
The statement:

temp2 = head;
and then free(temp2);
will make the head free and you won't be able to get next of head
as cheched in while condition.

• Reply • Share ›

Akhil • 7 months ago

```
void removingdublicate(node * head){
node *p=head;
node *q=NULL;
while(p->next!=NULL)
{
if(p->next->key==p->key)
{
if(p->next->next!=NULL)
p->next=p->next->next;
else
p->next=NULL;
}
else
p=p->next;
}
printf("After deletion\n");
node *s=head;
while(s!=NULL){
printf("%d",s->key);
s=s->next;
}
}
```

• Reply • Share ›

NancyD • 8 months ago

in the function removeDuplicates, in the while loop why cant we write while(current!=NULL) instead of while(current->next!=NULL)

1 • Reply • Share ›

Bewkoof_coder → NancyD • 7 months ago

bcz if current -> next is null ,that means the current node is the last node and their is no any node after it,so no need to go inside the loop as we reached the last node.

1 • Reply • Share ›

Saurabh • 8 months ago

http://ideone.com/iGvQCm

• Reply • Share ›

Shivanshu Chauhan • 8 months ago

void remove_duplicate(node * head){

node * a = head;

while(a->next != NULL){

while(a->key == a->next->key && a->next != NULL){

node * temp = a->next;

a->next = temp->next;

delete temp;

}

a = a->next;

}

}

What is the error in this ?????

• Reply • Share ›

DS+Algo Shivanshu Chauhan • 7 months ago

Consider 2->2->2->2->NULL

It goes in outer loop then it goes in inner loop,

then it comes out of inner loop as 2->NULL, with a pointing to 2.

a=a->next makes a pointing to null.

Now outer loop checks for a->next, but a is pointing to NULL.

Hope you understood otherwise feel free to ask.

Advice:

When you give a check for (a->next==NULL), always check for a==NULL too.

e.g.: instead of,

if(a->next==NULL),

always use if(a==NULL && a->next==NULL)

and instead of,

if(a->next->next==NULL),

always use if(a==NULL && a->next==NULL && a->next->next==NULL)

• Reply • Share ›

Saurabh ↪ Shivanshu Chauhan • 8 months ago

There is an error in your code.
while(a->key == a->next->key && a->next != NULL)
Lets say a->next is NULL then this while condition will generate run time error. You should change the condition as:
while(a->next != NULL && a->key == a->next->key)
In your case, if a->next is null then first a->next->key is accessed which is wrong after that the NULL condition is checked. So here is a small mistake. First check the NULL condition after that access the element.

• Reply • Share ›

Anand Namdev ↪ Saurabh • 8 months ago

No it wont , because before the condition while(a->key==a->next->key && a->next!=NULL) the coder has inserted a while loop just above it that while(a->next!=NULL) hence it will enter in the inner loop only when a->next!=NULL.

• Reply • Share ›

Saurabh ↪ Anand Namdev • 8 months ago

Oh.. my bad.. i did not notice that.

• Reply • Share ›

ravi • 9 months ago

if only 2 nodes r there then segmentation fault..
bcoz ur next_next points to some unknown location
so this code doesn't work for me...

• Reply • Share ›

Stack ↪ ravi • 9 months ago

It works fine say nodes are 1, 1
then next pointer will point to NULL
free current->next
then current->next=next_next(NULL).

Next time u try to run the loop current->next==NULL so it fails..

• Reply • Share ›

siddharth • 9 months ago

I'm getting TLE for this solution

• Reply • Share ›

Load more comments