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().

```
/*Program to remove duplicates from a sorted linked list */
#include<stdio.h>
#include<stlib.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);

  getchar();
}
```

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

References:
cslibrary.stanford.edu/105/LinkedListProblems.pdf

Related Topics:

- Clone a linked list with next and random pointer | Set 2
- Given a linked list of line segments, remove middle points
- Construct a Maximum Sum Linked List out of two Sorted Linked Lists having some Common nodes
- Given a linked list, reverse alternate nodes and append at the end
- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List

Writing code in comment? Please use **ideone.com** and share the link here.

Holmes Donalds • 23 days ago

/* Function to remove duplicates from a unsorted linked list */

```
void removeDuplicates(struct node *start)

{

struct node *ptr1, *ptr2, *dup;

ptr1 = start;

ptr2 = ptr1->next;

while((ptr1) && (ptr2))

{

if(((ptr1->data)^(ptr2->data)) == 0)

{

ptr2 = ptr2->next;
```

geekyprateek • a month ago

simple java function

```
void removeDup(LinkedList l){

Node n=l.head;

if(n==null)

return;

while( n!=null && n.next!=null ){

if(n.data!=n.next.data){

n=n.next;

}else

{

n.next=n.next.next;

}

}

}
```

• Reply • Share ›

surbhijain93 • a month ago

is this fine?

```
void remove1(struct node *head)

{

struct node* current=head;

struct node *temp;

while(current!=NULL && current->next!=NULL)

{

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

{
```

```
temp=current->next;

current->next=current->next->next;

free(temp);

}

else

current=current->next;

}
```

• Reply • Share ›

Ekta Goel ➚ surbhijain93 • a month ago

Seems correct, as the questions says the list is sorted in non-decreasing order, its fine for that purpose. However, if the question is varied little that the list is not in some particular order, it'll not work as you can see from your code.

1 • Reply • Share ›

munjal • a month ago

code : http://geekforgeekss.blogspot....
time complexity : O(n)

• Reply • Share ›

siddharth • a month ago

In this example, I tried it myself. The only thing I did different was in the while() if removeDuplicate() function, loop is gave the condition as
while(current != NULL), and that resulted in segmentation fault.

Why?
One of the reasons i think is that i am comparing last node of the LL with the next node i.e. NULL. Can that cause SegFault? If not then what is causing it?

• Reply • Share ›

Maneesh Rao • 2 months ago

Here is the Java Implementation

```
public class LinkedList {
Node head;
Node current;

public void RemoveDupes(){
current=this.head;
Node next=this.head.getNext();
```

```
while(next!=null)
if(current.getData()==next.getData()){
next=next.getNext();
}
else{
current.setNext(next);
current=next;
next=current.getNext();
}
```

see more

• Reply • Share ›

Shashank • 3 months ago

The code given below is a code written in
c++ . This code deletes all the duplicate contents of the linked-list .
In the code there is a pointer(ptr) of type node which points to a node. There is also a pointer pptr which points to the previous node of
ptr . Create function is used to insert elements in the linked-list by creating nodes . Display function is used to display the elements in the list . Let's talk about the delete_duplicate function , this function uses two loops both of which are while loops . First(outer) while loop is used to select an element using the temp pointer and set the values of ptr and pptr pointers . Then we check for duplicate content in the if clause of inner loop . The inner loop is used to compare all the other elements of the linked-list with the selected element . If the duplicate content is found then the particular node is deleted using the line

```
pptr->link=ptr->link;
ptr=ptr->link;
```

and the ptr is pointed to the next node without changing pptr . In the else

see more

• Reply • Share ›

Agam • 3 months ago

Recursive Implemetation :

```
void remove_duplicates_in_sorted(struct node *head)
{
struct node *temp;
if(head == NULL || head->next == NULL)
return;
```

```
while(head->data == head->next->data)
{
temp = head->next->next;
free(head->next);
head->next = temp;
}
remove_duplicates_in_sorted(head->next);
}
```

1 ^ | v • Reply • Share ›

Guest • 3 months ago

this can be done using 2 pointer no need of 3 pointer(including formal parameter)

```
void removeDuplicates(struct node *head)
{
struct node *dup;
if(head==NULL)
{
printf("list is empty");
return ;
}
while(head->next !=NULL)
{
if(head->data==head->next->data)
{
dup=head->next;
head->next=head->next->next;
free(dup);
}
else
head=head->next;
}
```

^ | v • Reply • Share ›

Chirag • 3 months ago

1 -> Traverse the list and check the node with the next node;
2 -> If value of both the node is same, use below algorithm:

```
if(temp->data == temp->next->data)
{
struct *delptr = temp->next;
temp->next = temp->next->next;
free(delptr);
}
```

[It saves the memory of extra pointer and few steps too.]

[It saves the memory of extra pointer and few steps too.]

• Reply • Share ›

Komal Singh ➚ Chirag • 2 months ago

why is the code not running when I declare delptr outside the if condition at the beginning of the function?

• Reply • Share ›

Novice_currently ➚ Komal Singh • 2 months ago

Post the code here..I can check

• Reply • Share ›

Ashish Jaiswal • 3 months ago

```
#include<stdio.h>

#include<stdlib.h>

struct node;

void push(struct node**,int);

void print(struct node*);

void removeduplicates(struct node*);

typedef struct node

{

int data;

struct node*next;

}Node;
```

see more

• Reply • Share ›

MKumar • 4 months ago

```
public void findDuplicateFromSorted(final Node node)
{
Node temp;
temp = node;
Node next_next = temp.next;
if(temp==null){
System.out.println("List is Empty");
}
if(temp.next==null){
```

```
if(temp.next==null){
System.out.println("Only one records exist, hence no Duplicate Found");
}
while (temp != null)
{
if (temp.key == next_next.key)
{
temp.next = next_next.next;
}
temp = temp.next;
next_next = next_next.next;
}
}
```

1 • Reply • Share ›

ecuses • 8 months ago

Why does it have to be sorted? I tested with unsorted lists and it seems to work ok.

• Reply • Share ›

Hinata Hyuga ecuses • 2 months ago

In this algo we are checking a node with its next node and not the whole list, In sorted list the node with same data will all be adjacent to each other and not otherwise.

• Reply • Share ›

Gaurav ecuses • 8 months ago

this method is only for sorted linked list. Reason being, in sorted LL same elements appear consecutively. Try with different test case of unsorted LL in this method eg: 2->2->4->3->2->5. you will see that the 2 between 3 and 5 is not removed.

3 • Reply • Share ›

vipinkaushal • 9 months ago

an other easy approach

```
void deleteDuplicate(node **header)
{
node *ptr1=*header,*ptr2;
int i;
while(ptr1!=NULL)
{
i=ptr1->data;
ptr2=ptr1->next;
while(ptr2!=NULL&&ptr2->data==i)ptr2=ptr2->next;
ptr1->next=ptr2;
ptr1=ptr1->next;
```

```
}
return;
}
```

2 • Reply • Share ›

rj vipinkaushal • 5 months ago

memory leak , sire !

1 • Reply • Share ›

GameInsta vipinkaushal • 7 months ago

I think this method doesn't free memory. o.O

1 • Reply • Share ›

Gitanshu behal • 9 months ago

I think this implementation is a bit easier to understand:

```
void removeDuplicates(struct node* head)

{

if(head==NULL || head->next==NULL)

return;

node *prev=head,*current=head->next;

while(current!=NULL)

{

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

{

prev->next=current->next;
```

see more

4 • Reply • Share ›

Abhishek Chauhan • 9 months ago

Using a two node window:

```
void remove_duplicate(node *head)
{
if (!head)
return;
node * w1 = head;
node * w2 = head->next;
```

```
node * w2 = head->next;

while (w2)
{
if (w1->data == w2->data)
{
w1->next = w2->next;

head = w2;
w2 = w2->next;
free(head);
}
else
{w1 = w1->next;
w2 = w2->next;

}
}
}
```

• Reply • Share ›

nobody • 9 months ago

Is this code correct?

```
struct node* cur=head;
struct node* cur_next=head->next;
while(cur!=NULL && cur->next!=NULL)
{
while(cur->data==cur_next->data)
{
struct node *temp=cur_next;
cur_next=cur_next->next;
free(temp);
}
cur->next=cur_next;
cur=cur_next;
cur_next=cur_next->next;
}
```

• Reply • Share ›

ashish jaiswal ➚ nobody • 3 months ago

Not working for case like 3->4->4->4->NULL

• Reply • Share ›

Gaurav Nara ➚ nobody • 9 months ago

but you're not deleting the node.. you're just shifting the next pointer?

• Reply • Share ›

SANTOSH KUMAR MISHRA • 9 months ago

```
void RemoveDuplicate(ListNode *head)
{
ListNode *current,*temp;
current = head;
if(!current)
{
printf("\n\nList is empty.\n\n");
return;
}
while(current->next->next)
{
if(current->data == current->next->data)
{
temp = current->next;
current->next = current->next->next;
free(temp);
continue;
}
```

see more

• Reply • Share ›

ANA • 9 months ago

please check it .

```
void remove_duplicate( struct node *first )

{

if ( first == NULL && first->next == NULL ){

return;

}

int cur = first->data;

struct node *pre = first;

struct node *current = first->next;

while ( current != NULL ) {

if ( cur == current->data) {
```

see more

• Reply • Share ›

popeye • 10 months ago

One can use a recursive version as well. http://ideone.com/4UeBkp
Time complexity : O(n)
Auxiliary space: O(1) (No stack space taken with tail call optimization)

• Reply • Share ›

ashish jaiswal ➚ popeye • 3 months ago

i am not able to understand your code...will you explain it?? while making recursive call i am not able to understand where it links previous node to node next to deleted node??

• Reply • Share ›

Aman • 10 months ago

Solution with less number of variables:

```
void removeDuplicates(Node* head)

{

while(head->next != NULL)

{

if(head->data == head->next->data)

{

Node* temp = head->next;

head->next = head->next->next;

temp = NULL;

}
```

see more

• Reply • Share ›

danny • a year ago

We can even use hash table to delete duplicate entries in linked list..
1.) We will map every value of linked list to hash table.
2.) While mapping we can compare if the two values hash to same location in hash table,then we get the address of the two values in linked list and delete one of them.
3.)Arrange pointers of the linked list .

Advantage:if there are millions and more elements and few of them are duplicated then it is much optimized then above solution.
Disadvantage:extra space for hash table implementation
@GeeksforGeeks please correct me if i am wrong...
1 • Reply • Share ›

Jonathan Chen ➚ danny • a year ago

How is it more optimized? I don't think it is.

The original strategy traverses the list only once. In order to remove duplicates in general, you have to look at every item at least once.

Your strategy goes through every element once as well. However, the downside is that you also use an auxiliary hashtable which is worse in terms of space complexity.

If there are millions more elements, your strategy has to traverse through every single one of them, just like the above approach does.
• Reply • Share ›

danny ➚ Jonathan Chen • a year ago

Yes, you are write It is not much optimized then above algorithm but it's another way of doing it and I even mentioned this disadvantage in my post.... but it can come in handy when linked list is not sorted .
2 • Reply • Share ›

Himanshu Dagar • a year ago

can refer to the below code also

http://ideone.com/BOzFhK
1 • Reply • Share ›

setu • a year ago

If there are millions of data in it and one number is duplicate somewhere in middle then how we will traverse it one by one then would take lot of time while compilation. please provide a solution for it
• Reply • Share ›

neo • a year ago

@GeeksforGeeks team the condition in the while statement should contain current!=null also because we are doing current=current->next in else part ,so if current is null then current->next in while condition will cause seg fault.please correct me if i am wrong
• Reply • Share ›

Jayanth ➚ neo • a year ago

I dont think it is necessary as when "current->next" itself is not null current cannot be null...So i dont think that check is necessary...in fact that check is implied...can u post some test cases where it might fail...may be i am not seeing something u hav...

• Reply • Share ›

Guest • a year ago

```
struct treeNode * removeDuplicateFrmOrderedLinkList(struct  treeNode* root)
{
    struct treeNode *next=root;
    struct treeNode *prev=NULL;

    while(next!=NULL)
    {
        prev=next;
        next=next->next;

        while(next!=NULL && next->val==prev->val)
        {
            struct treeNode *current=next;
            next=next->next;
            free(current);
        }
        prev->next=next;
    }

    return root;

}
```

• Reply • Share ›

Adarsh • 2 years ago

```
 #include <stdio.h>
#include <string.h>

typedef struct node
{
    int data;
    struct node *next;
}node;



void insert(node **head_ref, int val)
```

```
void insert(node **head_ref, int val)
{
    node *curr;
    node *temp;
    temp = (node *)malloc(sizeof(node));
    temp->data = val;
    temp->next = NULL;
    curr = *head_ref;
```

see more

• Reply • Share ›

ultimate_coder • 2 years ago

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

trilok ultimate_coder • 2 years ago

```
/* 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* dup_node;

/* 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 till duplicates exist*/
while( (current->next != NULL) && (current->data == current->next->data) )
{
/*The sequence of steps is important*/
dup_node = current->next;
current->next = dup_node->next;
free(dup_node);
}
current = current->next;
}
}
```

• Reply • Share ›

Ujjwal • 2 years ago

How about using 2 pointers to solve this..!!
-Both point to head initially.
-Advance 2nd pointer, if same data, go on advancing 2nd pointer until we get node with different data. Once we get that node, make 1st_node->next = 2nd_node;
1st_node = 2nd_node;

Repeat this till we reach the end of the list..
Correct me if i m wrong.. :)

1 • Reply • Share ›

Deepak ➦ Ujjwal • 2 years ago

Your code will remove duplicate nodes but it has wastage of space because you didn't free those nodes which have duplicate values.

1 • Reply • Share ›

Avinash Srivastava ➦ Deepak • 8 months ago

will work well in java because you have the garbage collecior doing the job for you..

• Reply • Share ›

Pranav • 2 years ago

```
 /* Paste your code here (You may delete these lines if not writing code) */
void removeDuplicates(struct node* head)
{
  /* Pointer to traverse the linked list */
  struct node* p = head;
  struct node* t=NULL;
  while(p)
  {
     t=p;
     if(!p->next)
        return;
     p=p->next;
     if( t->data == p->data )
     {
        t->next = p->next;
        free(p);
        p=t;
     }
  }
}
```

• Reply • Share ›

Sunny • 2 years ago

Similar code,but instead of next used prev.

[sourcecode language="C++"]
#include <iostream>
#include <stdlib.h>
using namespace std;

typedef struct link
{
int x;
struct link* next;
}node;

node* head;
void add(int a)
{
node* temp=new node;//(node*)malloc(sizeof(node));
temp->x=a;
temp->next=head;

see more

• Reply • Share ›

Kunaal Ahuja • 3 years ago

This code removes duplicates only if they occur in succession.Otherwise it won't work.

for eg.
12 2 7 9 12 2 9
for these set of inputs the code won't work

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

hARRY ➚ Kunaal Ahuja • 2 years ago

Hello read the problem statement carefully!!

• Reply • Share ›

Tim • 3 years ago

```
int RemoveDuplicates(linklist l){ //array as a helper to store the pointers
    linklist p = l->next;
    linklist a[100] = {0};
```

```
a[0] = p;
int i = 0;
int j = 0;

while(p != NULL){
      p = p->next;
      if (p == NULL) break;

      while(j >0 || j == 0 ){
            if(p->data == a[j]->data)
               break;
            j--;
      }

      if (j < 0){
```

see more

• Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Privacy

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Recent Comments

- Goku

  That would have higher time complexity.Method 1...

  Write a function to get the intersection point of two Linked Lists. · 7 minutes ago

- Guest

  calling the 2nd solution as dp is stretching...

Longest Even Length Substring such that Sum of First and Second Half is same · 21 minutes ago

- Goku

  They are considering 0 based indexing instead...

  Write a function to get Nth node in a Linked List · 1 hour ago

- lebron

  since the array size is 5, it takes constant...

  K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time) · 5 hours ago

- lebron

  merge sort

  K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time) · 5 hours ago

- Shubham Sharma

  You saved my time :)

  Searching for Patterns | Set 2 (KMP Algorithm) · 6 hours ago